Exposing and Explaining Misbehaviours of Deep Learning Systems

Doctoral Dissertation submitted to the

Faculty of Informatics of the *Università della Svizzera Italiana*

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

presented by

Tahereh Zohdinasab

under the supervision of

Prof. Paolo Tonella and Prof. Vincenzo Riccio

April 2024

Dissertation Committee

Dissertation accepted on 5 April 2024

[illegible]

Research Advisor

Prof. Paolo Tonella

[illegible]

Co-Advisor

Prof. Vincenzo Riccio

PhD Program Director

Prof. Walter Binder

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

[illegible]

Tahereh Zohdinasab
Lugano, 5 April 2024

To my beloved son

Abstract

Assessing the quality of Deep Learning (DL) systems is crucial, as they are increasingly adopted in safety-critical domains. Researchers have proposed several input generation techniques for DL systems. While such techniques can expose failures, they do not explain which features of the test inputs influenced the system's misbehaviour. This research delves into diverse methodologies aimed at overcoming challenges inherent in testing DL systems, with a particular focus on generating targeted test cases and interpreting system behaviours. To this aim, we proposed three novel testing approaches for DL systems, i.e., DEEPHYPERION-CS, DEEPATASH, and DEEPTHEIA.

DEEPHYPERION-CS explores the feature space at large using Illumination Search and provides a unique characterisation of a DL system's quality through an interpretable map which represents the highest-performing (i.e., misbehaving or closest to misbehaving) inputs in the space of the relevant, domain-specific features. We introduce a novel methodology to guide users in manually defining and quantifying feature dimensions effectively. Our empirical study shows that DEEPHYPERION-CS is more effective than state-of-the-art DL testing tools in generating failure-inducing inputs associated with highly diverse features.

DEEPATASH is a focused test generator, i.e., a solution for generating failure-inducing inputs with specific features. It can address the development to operation (dev2op) data shift phenomenon, by focusing on interesting feature values observed in operational environments. Further enhancing test generation efficiency, DEEPATASH-LR integrates a surrogate model into the process. Experimental results show that both DEEPATASH and DEEPATASH-LR are effective in generating focused test inputs and improving the quality of the original DL systems through fine tuning on data with the targeted features without regression.

DEEPTHEIA is a fully automated illumination-based test generator capable of autonomously extracting features and exploring the feature space using diffusion models. It overcomes the limitation of illumination-based approaches such as DEEPHYPERION, i.e. the need of human expert involvement for the definition of the features and the need of generative input models that can be mutated during the search process.

Finally, we provide a thorough comparison of explanatory techniques used to understand DL system misbehaviours, including our newly proposed feature maps, shedding light on both their comprehensibility and limitations. Our findings contribute significantly to advancing testing methodologies and enhancing the interpretability of the causes of DL misbehaviours.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Paolo Tonella, for his unwavering guidance, invaluable support, and continuous encouragement throughout my doctoral journey. His expertise, patience, and mentorship have been instrumental in shaping my research and academic growth.

I am also deeply thankful to my co-advisor, Prof. Vincenzo Riccio, for his insightful feedback, valuable insights, and scholarly guidance, which have greatly enriched my research experience and contributed to the success of this thesis.

I am indebted to my family for their boundless love, unwavering encouragement, and steadfast belief in my abilities. Their endless support and sacrifices have been my source of strength and inspiration.

I extend my heartfelt appreciation to my friends and colleagues for their encouragement, camaraderie, and intellectual exchange. Their collaboration and camaraderie have made this academic journey fulfilling and memorable.

Lastly, I would like to express my gratitude to all those who have contributed to this thesis in any form. Your support, encouragement, and guidance have been invaluable, and I am deeply grateful for your contributions.

This work was supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

Contents

Chapter 1

Introduction

Deep Learning (DL) has become an essential component of complex software systems, including autonomous vehicles and medical diagnosis systems. As a consequence, the problem of ensuring the dependability of DL systems is critical.

Unlike traditional software, in which developers explicitly program the system's behaviour, one peculiarity of DL systems is that they mimic the human ability to learn how to perform a task from training examples [95]. Therefore, it is essential to understand to what extent they can be trusted in response to the diversity of inputs they will process once deployed in the real world, as they could face scenarios that might be not sufficiently represented in the data from which they have learned [60].

Most importantly, when such underrepresented inputs are discovered by testing techniques, test results should be interpretable, as developers need to understand which features of the test inputs might have caused a system's misbehaviour (e.g., which characteristics of an input image make the system wrongly classify it or which features of a driving scenario make the system drive the autonomous vehicle off the road). In particular, we consider both structural features (i.e., characteristics of the input itself) and behavioural features (i.e., characteristics of the output of the DL system when exercised by the input).

Several test generation approaches have been proposed for automatically testing DL systems [118, 155]. Some of them aim to pragmatically expose the highest number of misbehaviours [1, 156, 40]. Other approaches, instead, are guided by *ad-hoc* test adequacy metrics, such as neuron coverage [112, 51, 143, 154, 143, 91, 24, 26] or surprise coverage [70, 68], since traditional code coverage metrics fail to measure whether DL systems have been adequately tested. These approaches are effective in triggering multiple misbehaviours, but their output cannot be directly used to explain the behaviour of the DL system under test. For instance, using neuron coverage reports, developers cannot easily understand why the DL system did not handle correctly the misbehaviour-inducing inputs.

We introduce a novel way to assess the quality of DL systems by automatically generating a large, diverse set of high-performing (i.e., misbehaving or near-misbehaving), but qualitatively different test inputs that provide developers with a human-interpretable picture of the system's quality. With our approach, developers can understand how different structural and behavioural features of the inputs combine to affect the system's performance. To this aim, we developed DEEPHYPERION-CS[1], an open-source automated test input generator for DL systems based on Illumination Search. It promotes the inputs with higher Contribution Score (CS) i.e. inputs that contributed more to feature space exploration during previous search iterations. Its output consists of *feature maps* representing the generated inputs along with their performance (i.e., closeness to exposing a misbehaviour), in the space of the relevant, domain- and problem-specific structural and behavioural features (i.e., the feature space). A feature map provides a representation of the feature space, which is defined by a set of N relevant dimensions of variation (i.e., the map axes, each corresponding to an input feature). In a feature map, test inputs are placed based on their feature values. These maps can provide insights into a test set, such as revealing feature value combinations associated with tests that triggered misbehaviours or indicating the likelihood of observing a misbehaviour for each feature combination. Our empirical results show that DEEPHYPERION-CS outperforms the state of the art by significantly improving the efficiency and the effectiveness in finding misbehaviour-inducing inputs and exploring the feature space. Moreover, we show that DEEPHYPERION-CS can help DL developers by characterising the deficiencies of the DL systems' training dataset and by providing new data to expand it.

During the testing phase, feature maps identify the regions within the feature space that lack sufficient coverage [13]. However, during operation, it is possible to encounter critical feature values that are under-represented in the train/test datasets used during development for which new and diverse scenarios need to be collected and manually labelled [49]. Therefore, testers need to find multiple misbehaviour-inducing test inputs associated with specific feature combinations. These additional inputs can be used to improve the quality of the DL system in production, by fine tuning it on such new data. To this aim, we present a novel approach, to generate misbehaviour-inducing inputs using user-defined feature values. Our approach enables focused testing by generating diverse inputs with critical features, stressing the system by finding failures even in non-critical regions or by generating new data reflecting underrepresented or unseen feature value. For instance, consider a scenario where a deployed DL system must handle feature combinations that were infrequently or never observed during development, known as the "dev2op" shift. To assess such feature combinations, we propose DEEPATASH, a search-based focused test generator for DL systems. DEEPATASH can be configured with various search strategies (single or multi-objective) and sparseness metrics. It takes target feature value ranges as input and optimizes both input diversity and proximity to the target feature values in the feature map. We assessed the performance of DEEPATASH across two distinct classification tasks: recognizing handwritten digits and conducting

[1]This tool is based on DEEPHYPERION that is also part of this thesis work.

sentiment analysis on movie reviews. Our analysis revealed that DEEPATASH is effective in generating a wide range of test inputs capable of inducing failures within the target feature map cell across various usage scenarios.

We further extend DEEPATASH ś applicability to the autonomous driving system (ADS) domain. Testing ADSs is recognized as resource-intensive due to the need for costly executions, such as simulations, to evaluate the system behavior. As a result, DEEPATASH may allocate a significant portion of its test generation time to explore less promising areas of the feature space. These areas may either lack the target features or have minimal potential for triggering misbehaviours, as each execution (e.g., simulation in a candidate test scenario) consumes a considerable portion of the available budget. We present DEEPATASH-LR, a novel tool that efficiently performs focused test generation by leveraging a surrogate model. DEEPATASH-LR employs a surrogate model as a proxy for actual system's execution, thus sidestepping the need for resource-intensive evaluations, which would involve complete simulations of the driving tasks on the candidate test scenarios. Our empirical findings underscore the indispensability of the surrogate model in producing inputs that induce misbehaviour within predefined targets. Furthermore, the inputs generated by DEEPATASH-LR have proven instrumental in fine-tuning the ADS under test, thereby enhancing its performance on previously overlooked feature combinations, particularly those inducing failures.

One significant challenge encountered with illumination search algorithms lies in the definition of features, typically tailored to specific problems and domains, which requires the collaboration of multiple domain experts tasked with identifying features (referred to as map dimensions) and devising metrics for their quantification. Furthermore, human input is also essential for crafting models of the input subject to perturbation by mutation operators. While this human involvement enhances the interpretability and relevance of feature dimensions, defining features, metrics, and input models by human experts poses non-trivial challenges that potentially limits the broad applicability of this testing methodologies. Introducing a novel approach named DEEPTHEIA, we address the limitations of the current state of the art. DEEPTHEIA incorporates automated feature extraction and input perturbation operators, leveraging modern generative DL techniques. In particular, DEEPTHEIA utilizes the knowledge automatically acquired by a DNN about the target domain to extract features that effectively capture the primary characteristics of test inputs and uses them as feature dimensions for illumination search. Our empirical evaluation on two different image classification tasks, hand written digit classifier and classification of real-world images, shows that DEEPTHEIA generates feature maps with exceptional discriminative capabilities, enabling the identification of feature value combinations that induce misbehaviors in the DL system. Moreover, our human study results indicate that the automatically extracted features yield cohesive groups of inputs, all mapped to the same cell. This observation suggests a certain level of human interpretability for each feature map cell.

Recent research has proposed several techniques for explaining DL (mis)behaviour,

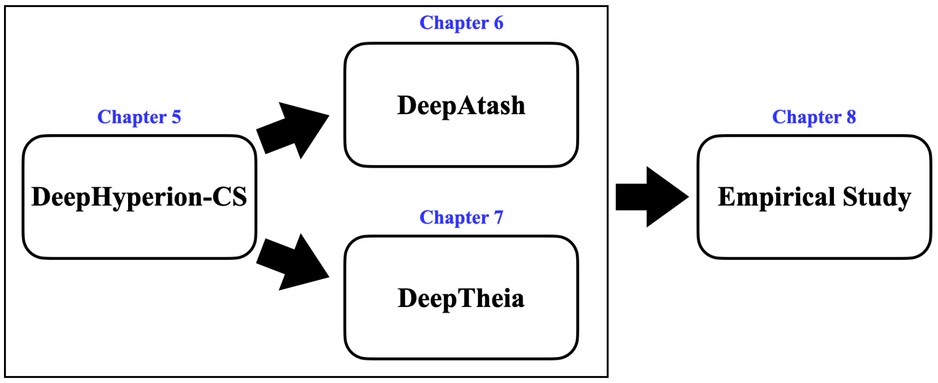


Figure 1.1. Thesis contributions structure.

using different strategies and producing mainly low-level input explanations (i.e. raw input elements). These explanation methods identify a part of the input vector that is considered relevant for a specific DL prediction. In contrast, our approach provides developers with explanations based on high-level features of the input (i.e. manually or automatically defined abstractions of the input). To compare the similarity between the output of different explanatory techniques and to assess to what extent they are understandable by Software Engineering (SE) experts, we conducted an empirical study involving our approach and 2 state-of-the-art techniques for DL explanation in 13 configurations, applied to 2 different DL classification tasks. We have also collected answers from 48 questionnaires submitted to human assessors for comparing the understandability of explanations for DNN misbehaviours. Experimental results show that low-level and high-level techniques provide, for the same misbehaviour-inducing inputs, dissimilar yet highly complementary explanations. So, SE experts should consider both granularity levels. On the other hand, our results indicate that further research is needed to produce better explanations, since experts deemed none of the explanations as useful in 28% of the cases.

Figure 1.1 shows an overview of this thesis. The contributions of this thesis are as follows:

- DeepHyperion-CS, a search-based testing approach for DL systems that explores at large, the feature space, by providing developers with an interpretable feature map.

- DeepAtash, the first search-based focused testing approach for DL systems that generates inputs with target features.

- DeepTheia, fully automated illumination-based test generator that autonomously extracts features and explores the feature space using cutting-edge diffusion models.

- An empirical study for in-depth comparison of explanatory techniques for DL misbehaviours.

Publications

1. Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. "Deep-hyperion: exploring the feature space of deep learning-based systems through illumination search." In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21).
2. Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. "Efficient and effective feature space exploration for testing deep learning systems." ACM Transactions on Software Engineering and Methodology 32, no. 2 (2023): 1-38.
3. Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. "DeepAtash: Focused Test Generation for Deep Learning Systems." In Proceedings of the 32th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23).
4. Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. "An Empirical Study on Low-and High-Level Explanations of Deep Learning Misbehaviours." In 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '23).
5. Tahereh Zohdinasab, and Andrea Doreste, "DeepHyperion-UAV at the SBFT Tool Competition 2024 - CPS-UAV Test Case Generation Track." In the proceedings of the 2024 ACM/IEEE International Workshop on Search-Based and Fuzz Testing (SBFT '24).
6. Tahereh Zohdinasab, Vincenzo Riccio, and Paolo Tonella. "Focused Test Generation for Autonomous Driving Systems." To be published in ACM Transactions on Software Engineering and Methodology.

Chapter 2

Background

2.1 Illumination Search

Evolutionary algorithms are a family of meta-heuristic optimization algorithms that evolve a *population* of *individuals* (i.e. a set of candidate solutions to an optimization problem) by means of genetic operators such as *mutation* and *crossover*. A *fitness function* provides an approximate, heuristic distance of each individual from the searched optimum. During evolution, the best individuals are selected for the next population based on the values of one or multiple fitness functions. The solutions found by evolutionary algorithms might be concentrated in a small portion of the input space, especially when the search landscape includes local optima with a large basin of attraction.

Novelty search algorithms are a family of algorithms designed to find solutions spread across the entire input space: they reward individuals that exhibit diversity of behaviours, instead of promoting only those that contribute to progress toward the optimum [84, 97]. Evolutionary algorithms can be effectively combined with *novelty search* to mitigate the problem of local optima since they trade off a lower pressure toward optimal fitness values with a higher diversity in the population being evolved [119].

Illumination Search denotes a family of search algorithms that "illuminate" the input space, i.e., find the best solution at each point within all regions of the search space. These algorithms effectively balance *exploitation*, i.e., the mechanisms that reward the most promising inputs, with *exploration*, which allows to explore the search space at large by promoting input diversity. Illumination Search has been already effectively used for testing DL systems. Specifically, Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [101] is a popular illumination search algorithm [153]. Figure 2.1 illustrates the main loop of this algorithm.

MAP-Elites starts by filling an empty N-dimensional feature map with an initial population to be evolved. Its evolutionary search process continues until the termination of execution budget. In each iteration, MAP-Elites selects an individual occupying a cell of the current map and mutates it to generate a new individual. To determine the

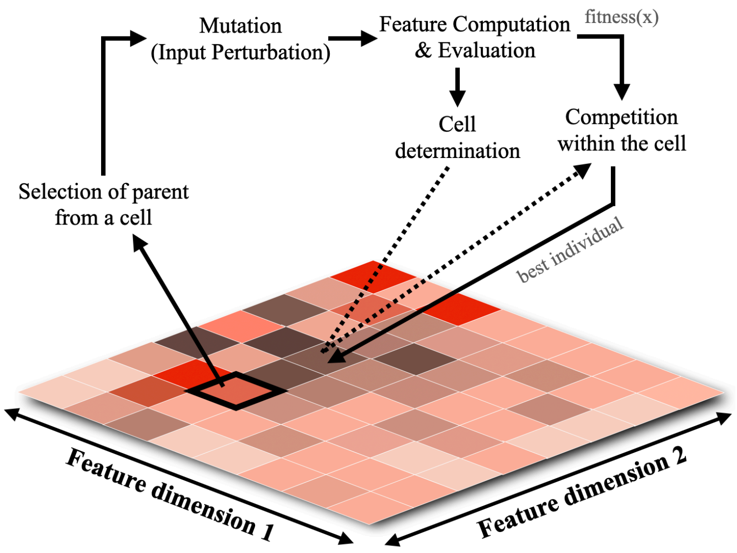


Figure 2.1. Overview of the MAP-Elites algorithm.

cell corresponding to the new individual, MAP-Elites computes its feature values. If the identified map cell already contains an individual, MAP-Elites places the individual with the higher fitness value in the map, thus performing a local competition. When the termination condition is satisfied, the algorithm outputs the feature map containing the highest fitness individuals.

2.2 Explainable AI

Explainable Artificial Intelligence (XAI) encompasses a set of methods designed to assist humans in interpreting decisions and understanding the behaviour of AI systems, by generating explanations. In the field of XAI, a variety of approaches are available to explain the predictions of DNNs, since DL architectures are widespread and particularly difficult to explain in terms of their input elements [115, 127, 139, 130, 89, 100]. We refer to these approaches as low-level explanation techniques, as they identify a subset of the input vector that is considered relevant for a given DL prediction.

We categoris low-level explanatory techniques into two groups based on the granularity of input elements provided in their explanations: (1) fine-grained and (2) coarse-grained. While the former provides explanations that highlight the small, atomic elements of the input (e.g. the pixels of an image input or words of a text input) which contribute to the final prediction; the explanations by the latter group indicate relevant contiguous regions of input elements, e.g., sets of contiguous pixels. Integrated Gradients (IG) [139] and LIME [115] represent the state of the art in XAI, respectively in the fine-grained and coarse-grained categories.

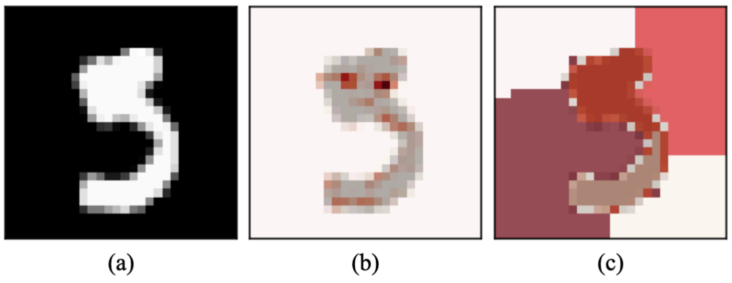
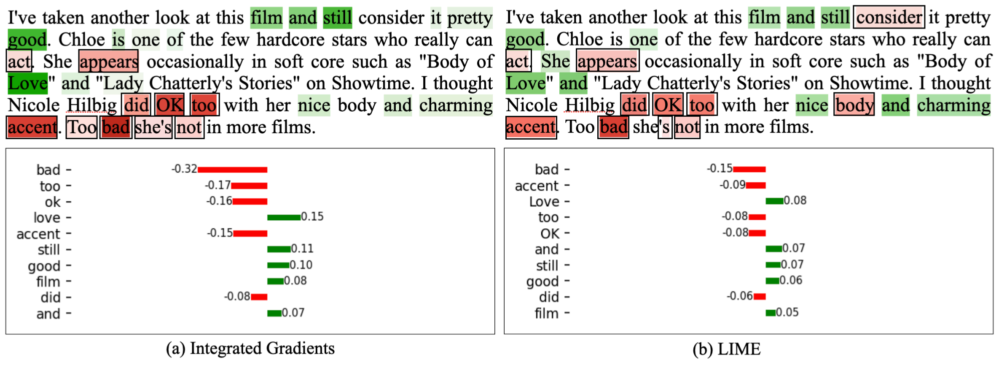


Figure 2.2. (a) A sample mis-classified hand written digit 5; (b) Explanation by Integrated Gradients; (c) Explanation by LIME. (darker red highlights indicates more contributing to the model's output)


I've taken another look at this film and still consider it pretty good. Chloe is one of the few hardcore stars who really can act. She appears occasionally in soft core such as "Body of Love" and "Lady Chatterly's Stories" on Showtime. I thought Nicole Hilbig did OK too with her nice body and charming accent. Too bad she's not in more films.

I've taken another look at this film and still consider it pretty good. Chloe is one of the few hardcore stars who really can act. She appears occasionally in soft core such as "Body of Love" and "Lady Chatterly's Stories" on Showtime. I thought Nicole Hilbig did OK too with her nice body and charming accent. Too bad she's not in more films.

Figure 2.3. A sample mis-classified positive movie review: (a) Explanations by Integrated Gradients; (b) Explanations by LIME. In the top explanation visualization, the darker the red/green highlighting, the larger is the contribution of the corresponding word to the negative/positive sentiment. The words contributing to the negative sentiment (i.e., the wrong class) are marked with a black square. The explanation visualizations at the bottom report red/green bars for the words mostly contributing (top 10) to the negative/positive sentiment. The bar length and the attribution value indicate how much each word is contributing to the corresponding sentiment.

2.2.1 Integrated Gradients

Associating the prediction of a DNN to the input elements that caused such prediction (aka, *input attribution*) is a way of explaining the model's behaviour. For instance, in an image classification model, an attribution method could reveal which pixels of the image are responsible for a certain label being predicted.

The gradient of the output with respect to the input elements (e.g., pixels) quantifies the impact of each input element on the prediction of a DNN, therefore the gradient computation can be considered as the most basic attribution method.

Sundararajan et al. [139] took an axiomatic approach for attributing prediction of DNN to its input features , called INTEGRATED GRADIENTS. They introduced two axioms to be preserved, *Sensitivity* and *Implementation Invariance*, and proposed INTEGRATED GRADIENTS by considering the straight-line path in DNN from the baseline (for example, black image in image classification task) to the input and cumulating the gradients at all points along the path for a given number of steps. The *number of steps* is a hyperparameter of this technique, representing the steps needed by INTEGRATED GRADIENTS in the gradient approximation for each input image. The output of this technique is a heatmap highlighting the important elements in the input (e.g. pixels of an image or words in a text). INTEGRATED GRADIENTS method can be implemented to generate explanation for a batch of inputs with predefined size.

Figure 2.2 (b) shows a heatmap generated by INTEGRATED GRADIENTS for an image classification task. The highlighted pixels are contributing the most to the output of the model. Figure 2.3 (a) shows an example of explanations generated by INTEGRATED GRADIENTS for a positive movie review, with the words contributing to the negative sentiment highlighted in red shades, while the green color highlights the words contributing to the positive sentiment (i.e., the darker the shade, the most important is the element to the final decision).

2.2.2 LIME

Local Interpretable Model-agnostic Explanations (LIME) proposed by Ribeiro et al. [115] is a method that provides a low-level explanation for the predictions of any classification model by identifying the portions of the input that mostly affect the model's output. LIME produces an explanation for each input instance by mimicking the behaviour of the model in the neighbourhood of the input. Actually, LIME perturbs the input to train a linear model (e.g. a decision tree) around the input itself and generates a predefined number of samples, while the model under test itself is treated as a black box. The number of samples can be customized, as it can be set as a hyperparameter of the technique. Such perturbed input regions are called *super-pixels* and their impact on the linear model's decision determines their importance in the final explanation.

LIME is applicable to both image and textual data and provides a visual representation of the generated explanations. It has been used in different studies to understand the reasons behind DNN decisions [83, 94, 46].

Figure 2.2 (c) shows an example of visualised explanation generated by LIME for a hand written digit 5: the darker is the color with which super-pixels are highlighted, the more important the super-pixel is towards the final decision. Therefore, the darkest super-pixels are the ones that are most likely responsible for the model's misbehaviour. Figure 2.3 (b) shows an example of visualised explanation generated by LIME for a positive movie review, with the words highlighted in red shades contributing to the negative sentiment and green shades contributing to the positive sentiment.

Chapter 3

Case Studies

Given the multitude of DL-based systems encompassing classifiers and regressors, where DL can be employed either directly or as an integral component within a complex system, it is essential to consider a diverse range of subject systems to capture this variability. We evaluated our approaches on different DL systems widely used in the literature to assess testing techniques for DL systems [118, 155].

3.1 Handwritten Digit Classifier

MNIST (Modified National Institute of Standards and Technology) is a large handwritten digit dataset widely used for machine learning research [118]. The data set includes 60,000 images for training and 10,000 images for testing. Figure 3.1 presents some sample digit images from MNIST data set.

The DL system recognises handwritten digits from the MNIST dataset [82]; hence, it performs a classification task. Its DNN predicts which digit is represented in a greyscale image. In particular, we consider the popular convolutional DNN architecture provided by Keras [22]. We trained this DNN on the MNIST training set using its default configuration, i.e., 12 epochs, batches of size 128, and a learning rate equal to 1.0. Our digit classifier achieved 99.8% classification accuracy on the MNIST testing set.

3.2 Self-driving Car

The popularity of self-driving cars has surged as advancements in autonomous driving, emphasizing the need for rigorous testing. Testing is vital to ensure the safety and adaptability of autonomous driving systems in diverse real-world scenarios. Simulators play a crucial role in this process, offering a controlled environment for efficient and reproducible testing. The BeamNG system is a simulation-based self-driving car. It implements an end-to-end, vision-based driving agent that can follow the lane in a road. BeamNG includes a DL-based Lane Keeping Assist System (LKAS), i.e., a DNN able to

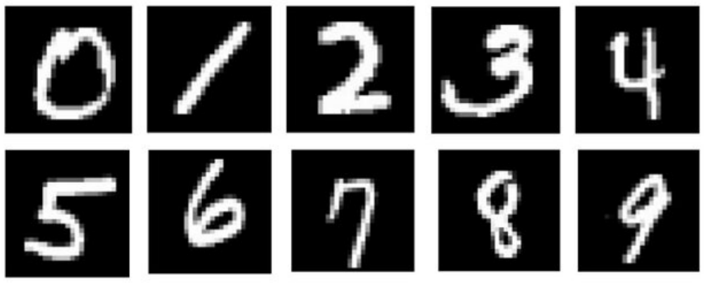


Figure 3.1. Sample digit images [18] from MNIST data set.

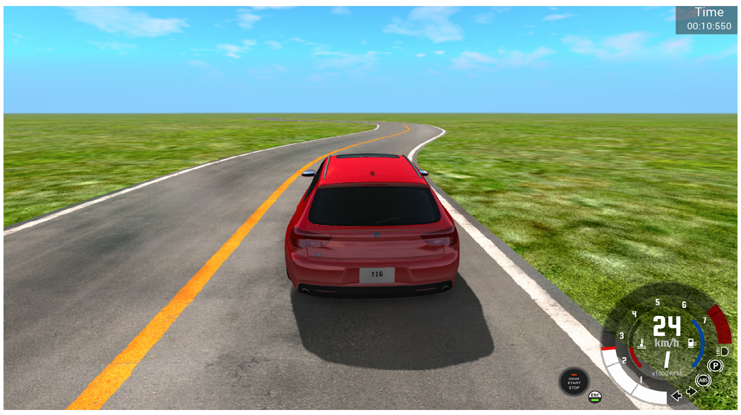


Figure 3.2. Test execution within the BeamNG simulator.

predict the steering angle of the car given the image of its onboard cameras; hence, it solves a regression problem. For this task, we adopted the widely known DAVE-2 architecture designed by Bojarski et al. at NVIDIA [17]. DAVE-2 operates through behavioural cloning, i.e., it learns to establish a mapping between images and steering angles based on examples.

The whole DL system is tested in the BeamNG.research driving simulator [10], a state-of-the-art simulator widely used in research [110]. We trained the model for 200 epochs, with batches of size 128 and a learning rate equal to 0.001, achieving a mean squared error (MSE) of $4.31e^{-5}$ on the test set. Our training set consists of images captured by the on-board camera, labelled with the steering angles provided by the simulator's autopilot while driving on virtual roads up to $25Km/h$.

The driving model's performance can be evaluated offline or online. Offline evaluation entails testing the DL model with pre-collected data in a non-real-time setting. The DNN's predictions are compared to ground truth labels, serving as an oracle, and a test is deemed unsuccessful if the error exceeds a predefined threshold. Online evaluation involves deploying the DL model in a real-time, interactive environment, often using a simulator. While the DNN still analyzes a stream of unlabeled driving images from an onboard camera, its predictions directly influence the overall system behavior, impacting subsequent driving decisions. Consequently, individual DNN prediction errors become less meaningful and computationally challenging, as there's no clear association of ground truth labels with incoming data.

3.3 Movie Sentiment Analysis

Text classification stands as a fundamental task in natural language processing, finding diverse applications such as sentiment analysis, topic labeling, spam detection, and intent detection. The Movie Reviews dataset [92] is a binary sentiment analysis dataset consisting of $50,000$ reviews from the Internet Movie Database (IMDb) labeled as positive or negative. The DNN solves a text classification problem: it predicts whether the review has positive or negative sentiment. We used a convolutional DNN architecture with an embedding layer provided by Keras [107], which accepts as input tokenised (with vocabulary size equals to 10K) and padded text with length limited to 2K words. We trained the model on the IMDB training set with 10 epochs, batches of size 32 with early stopping, and the *Adam* optimizer, achieving 88.19% test accuracy.

3.4 Image Classification

Image recognition is used to perform many machine-based visual tasks, such as labeling the content of images with meta tags, conducting content-based image searches, and providing guidance to autonomous robots. IMAGENET is a large and extremely popular image dataset, which has been used for the IMAGENET Large Scale Visual Recognition Challenge (ILSVRC) [126]. Figure 3.3 presents some sample images from different classes of IMAGENET data set. The large volume of annotated complex images allowed researchers to train deep neural networks, which require vast amounts of data to learn effectively. This dataset includes images of 1000 classes, partitioned into three sets: training (1.3M images), validation (50K images), and testing (100K images with held-out class labels). For this dataset, we used the pre-trained $ResNet50$ [57] neural network provided by the timm library[1] in Pytorch[2] with 81.17% accuracy.

[1]https://timm.fast.ai
[2]https://pytorch.org

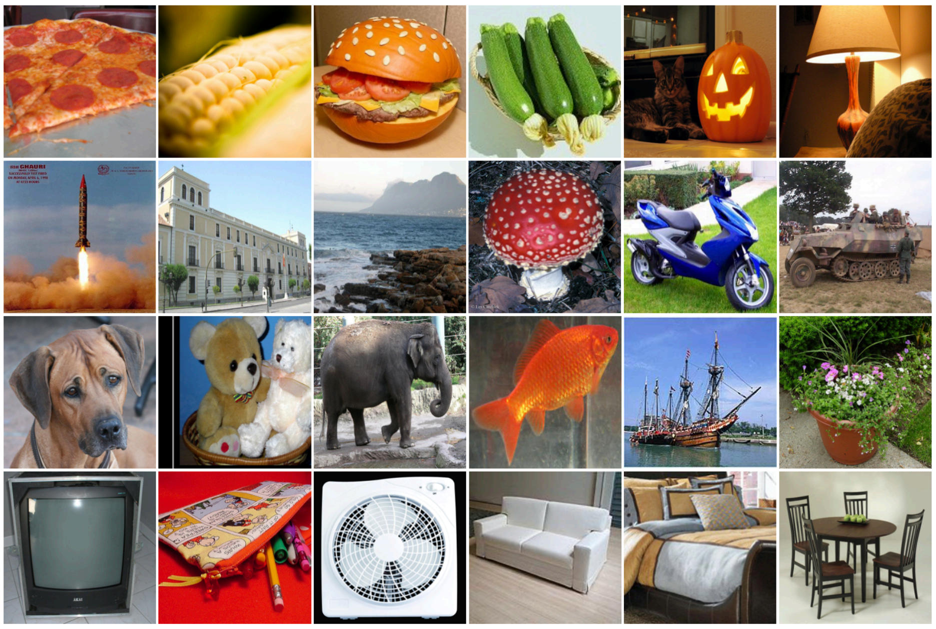


Figure 3.3. Sample images from ImageNet data set.

Chapter 4

State of the Art

4.1 DL Test Input Generation and Adequacy

Traditional test adequacy criteria, e.g., code coverage, cannot assess if DL systems are adequately exercised by a test set , as any test set, even a very weak one, is likely to cover the code that creates and uses a DNN fully, without however exercising its actual behaviours. DL systems' behaviour mostly depends on their training data, architecture and the tuning of several hyper-parameters, rather than the code. Recent research defined ad-hoc test adequacy metrics for DL systems and proposed input generation techniques guided by these metrics.

Pei et al. [112] defined neuron coverage, an adequacy criterion that measures the percentage of neurons whose activation level exceeds a certain threshold during testing. They also designed DeepXplore, a test generator guided by neuron coverage to detect behaviour inconsistencies between different DNNs. In particular, DeepXplore solves a joint optimization problem that maximizes both neuron coverage and differential behaviours. They calculate the gradients of the neurons in both the output and hidden layers with the input value as a variable and weight parameters as constants. They perform an iterative gradient ascent to adjust the test input, aiming to maximize the objective function of the joint optimization problem. However, its implementation necessitates multiple DL systems with comparable functionality to serve as cross-referencing oracles, a process that is hindered by scalability challenges and the difficulty of identifying similar DL systems.

Several other approaches extended neuron coverage [91, 90] or used it to drive test generation [51, 143, 91, 154, 24, 26]. DLFuzz [51] is a test input generator guided by neuron coverage. DLFuzz employs an iterative process where it selects the neurons to activate, aiming to increase neuron coverage. It introduces subtle perturbations to existing test inputs to guide DL systems in revealing incorrect behaviours. Throughout the mutation process, DLFuzz keeps mutated inputs that contribute to a specific increase in neuron coverage for subsequent fuzzing rounds. It identifies the erroneous behaviors when the prediction of the mutated input is different from the one produced by the original input.

DeepTest [143] maximises the neuron coverage of a DNN-based steering angle predictor by applying different image transformations to images captured by the on-board camera of an autonomous car. In particular, it adopts a greedy search algorithm to identify the combinations of the transformations that lead to higher neuron coverage and finds errorneous behaviours using domain-specific metamorphic relations. For instance, it checks whether an autonomous driving system behaves similarly when the input image is transformed into the same scene under a different weather condition.

DeepGauge [91] provides a set of multi-granularity coverage criteria that extend neuron coverage by taking the distribution of training data into consideration. For instance, k-multisection Neuron Coverage (KMNC) partitions the major function region of neurons into k sections and requires each of them to be covered by the test inputs. Neuron Boundary Coverage (NBC) and Strong Neuron Activation Coverage (SNAC) instead deal with corner case regions: they measure how many corner cases have been covered with respect to both boundary values and only upper boundary value of neuron activation functions, respectively. Ma et al. [91] also proposed coverage criteria at layer-level based on number of neurons that have once been the most active k neurons on each layer (Top-k neuron coverage). For test generation, they use multiple adversarial data generation algorithms i.e., Fast Gradient Sign Method (FGSM) [43], Basic Iterative Method (BIM) [77], Jacobian-based Saliency Map Attack (JSMA) [111] and Carlini/Wagner attack (CW) [19] and they show their coverage criteria can indicate a higher chance of detecting DNN failures.

DeepCT [90] uses a set of combinatorial testing (CT) criteria for DNNs based on the interactions between neurons. It evaluates the activation status of neurons in each layer by checking the proportion of interactions among activated neurons within a layer. This technique initializes a CT coverage table for the DNN and iteratively generates tests guided by the coverage criteria using random testing. Tests are generated layer by layer, targeting specific coverage goals such as t-way combination sparse coverage which quantifies the percentage of t-way neuron combinations in layer i where all neuron-activation configurations are covered by the test set.

DeepHunter [154] leverages multiple previously proposed coverage criteria [112, 91] as feedback to guide test generation. In particular, it keeps the seeds in the generation loop only if they contribute to the coverage metrics. To balance diversity, DeepHunter adopts a seed selection strategy which prioritizes new tests over the seeds that have been fuzzed many times. Their metamorphic mutation operator includes pixel and affine transformations. While the former modifies image contrast, brightness, blur and noise, the latter applies modifications such as scaling, shearing and rotation.

The test generators mentioned above generate adversarial inputs by adding small perturbations to the original inputs. Adversarial input perturbations are widely used by the machine learning community to affect model's predictions [16]. While adversarial attacks expose security and robustness vulnerabilities, they do not promote the spread of such attacks in the feature space. We aim to adopt model-based generative approaches which modify the input model parameters to ensure at the same time more diversity and more control on the realism of the generated inputs. For more complex inputs when the input model is not available or difficult to define, we use generative DL models.

Kim et al. [70] note that NC and KMNC lack practical utility as they provide limited information about individual inputs. They argue that a higher NC does not necessarily indicate a better input, as some inputs naturally activate more neurons. To overcome these limitations, they designed an adequacy criterion named Surprise Adequacy (SA) that measures the degree of “surprise” of test inputs with respect to the training set. They propose two variants of SA: (1) Likelihood-based SA (LSA) that estimates the likelihood of the system encountering a similar input during training using kernel density estimation; (2) Distance-based SA (DSA) that is calculated based on the Euclidean distance between vectors representing the neuron activation traces of the given input and the training data. In their work, SA was used for test case selection and retraining, not for test input generation.

Kang et. al [66, 68] introduced SINVAD for testing DL systems using the latent space of VAEs. SINVAD performs optimization to find inputs close to the decision boundary of the DL system. In particular, it adds perturbations to the elements of the input latent representations to generate surprising or misbehaviour-inducing inputs. To this aim, they employ either hill climbing or a genetic algorithm: hill climbing involves searching for images with a specified Surprise Adequacy [70], while for generating images that mimic one category but are classified as a different category, as in adversarial examples, they use the genetic algorithm. Their approach is dependent on the VAE for the definition of the search space, which may or may not be available for the DL system under test.

Another adequacy metric is mutation adequacy criteria that assess the ability of test data to expose artificially injected faults that simulate real faults (i.e., mutations). A notable work in this area is DeepCrime [61], a mutation tool built on top of the notion of statistical killing proposed by Jahangirova and Tonella [64], which injects mutations resembling real DL fault classes, such as the ones defined in the taxonomy by Humbatova et al. [60]. DeepMetis [117] is a search-based test generator that prioritizes mutation adequacy as its guiding principle. Fundamentally, a mutant is considered "killed" if the DL model under test exhibits correct behavior, while a "misbehavior" is observed upon evaluating its mutated counterpart. Through this approach, they generate new inputs that effectively target and kill mutants not killed by the original test set, hence, increasing the mutation killing ability of the test set.

All approaches mentioned above aim at maximising some internal adequacy metric, such as neuron or surprise coverage, while our aim is to explore the feature space of DL system and explain the misbehaviour of DL system through interpretable features. Moreover, current approaches necessitate white-box access to the activation levels of the DNN, particularly if they are guided by coverage criteria like neuron coverage. We are interested in testing at functional level the diversified feature combinations that trigger misbehaviours.

Other testing approaches attempt to maximise the number of failures by using a variety of techniques. AsFault [40] generates challenging scenarios that maximize the number of exposed system failures. Its primary objective is to thoroughly test ADSs by subjecting them to various difficult situations that they may encounter in real-world driving conditions. To this aim, it combines procedural content generation (PCG) and search-based testing (SBST). The procedural generation of road networks aims to provide a detailed description of road and lane structures using splines. Initially, roads are constructed independently by generating their segments. Subsequently, roads are combined on the same map to form complete road networks. As a search-based approach, AsFault employs a genetic algorithm to iteratively evolve road networks, with a fitness function which rewards those tests that cause the ego-car to move the farthest away from the lane centre.

NSGAII-DT [2] is a test generator designed for vision-based control systems. It is guided by decision trees that are constructed based on critical combinations of structural features learned throughout the exploration process. NSGAII-DT uses ranges of input/environment variables (e.g., pedestrian position/speed) to automatically identify the most critical regions of the input space. However, this approach is limited, as input/environment variables do not fully characterise higher level abstract properties of the road as well as behavioral properties of the driving system. Moreover, in their approach the user cannot specify a specific range of interest, as the approach automatically focuses on input values that most likely trigger failures.

DeepJanus [119] characterises a DL system's quality as its frontier of behaviours, i.e., pairs of similar inputs that trigger different (correct vs failing) behaviours of the system. It uses multi-objective search-based algorithm to generate diverse test inputs at frontier of the system. While DeepJanus provides users with a set of system's frontier inputs, it does not explicitly characterise them based on structural or behavioural features.

X. Zhang et al. [157] proposed an approach for generating test inputs with diverse uncertainty patterns (i.e., prediction confidence score and variation ratio). They do not define adequacy criteria, but they suggest to generate test inputs that target the least frequently covered uncertainty patterns. They introduced KuK, a tool that generates uncommon data using genetic algorithm. They define fitness function such that it prompts uncovered uncertainty types such as, data samples with high predication confidence score (PCS) and high Variation Ratio in terms of original prediction (VRO), while their mutation operator perturbs seeds only by adding random white noise to them.

AMBIEGEN [62] is a search-based framework for generating diverse misbehaviour-inducing test scenarios for ADSs. It explicitly promotes test diversity and employs a simplified system model to approximate results without running time-consuming simulations. It uses the NSGAII algorithm with two objectives: one evaluates the system's behaviour deviation from the expected, and the other measures test case diversity using a reference test case.

SAMOTA (Surrogate-Assisted Many-Objective Testing Approach) [52] is a testing approach that combines many-objective search and surrogate-assisted optimization techniques. It aims to achieve multiple safety objectives within a limited time budget while efficiently searching for critical test data using surrogate models that mimic the simulator but are computationally less expensive. Its approach consists of two search phases, global search using global surrogate models to explore the search space and capture the global fitness landscape, and local search using local surrogate models to exploit promising areas found by the global search. Each objective of SAMOTA is a safety violation, and the specific features of the misbehaviour-inducing inputs are not relevant for SAMOTA.

All the test generators we mentioned above aim at exposing misbehaviours. However, merely exposing DL misbehaviours is not sufficient to understand the input features causing them and, thus, thoroughly evaluate the system quality. In fact, a DNN model is commonly perceived as a black-box, and despite its exceptional performance, it often struggles to offer meaningful explanations for specific predictions or decisions [50, 141].

4.2 Explainable AI for DL Testing

In the literature, low-level explanations have been leveraged to interpret the output of DL testing.

Fahmy et al. [30] proposed HUDD (Heatmap-based Unsupervised Debugging of DNNs) to identify root causes of DNN failures. HUDD uses heatmap explanations generated by the Layerwise Relevance Propagation (LRP) technique and then clusters inputs with similar heatmap characteristics. They showed that by inspecting explanations generated from failure-inducing images within the same cluster developers can find common root causes for failures. HUDD has been used in the test generator SEDE (Simulator-based Explanations for DNN FailurEs) [31], to derive properties of unsafe images relying on a set of decision rules. SEDE generates images similar to the failure-inducing ones extracted by HUDD (i.e., mapped to the same clusters) and improves the DL model by retraining it with the artificially generated inputs.

Stocco el. al [134] proposed ThirdEye, a monitor for autonomous driving systems (ADSs) which relies on heatmaps produced by the SmoothGrad [132] technique to predict unsafe conditions in advance. ThirdEye computes confidence scores based on which parts of the image taken from the on-board camera contribute the most to the ADS decision and warns the main driving component of potential safety-critical failures, when the confidence is lower than a predefined corresponding threshold.

Existing works considered low-level explanations for testing DL systems. Instead, our goal is to consider different explanation levels (i.e., both low-level and high-level) and compare them to assess their usefulness and understandability in explaining DL misbehaviours.

4.3 Feature Extraction for DL Testing

Different techniques proposed in the literature for testing DL systems focus on the features of the test inputs.

O'Shaughnessy et. al [108] generate post-hoc causal explanations for black-box classifiers by leveraging a learned low-dimensional representation of the data. Their method involves constructing a generative model (e.g. a VAE) with a disentangled representation of the data and a mapping to the data space. They use a structural causal model (SCM) to formalize the relationships between independent latent factors, classifier inputs, and outputs. Our approach also relies on latent features of the input that are automatically extracted. However, our aim is different, i.e., to generate test inputs by covering the feature space while providing discriminative feature maps for further analysis of the model's behaviour.

Dola et. al [27] extracted feature vectors using a VAE trained on the training data of the DNN under test. These feature vectors establish a coverage domain for the application of Combinatorial Interaction Testing (CIT) on a partitioned latent space, facilitating the measurement of test coverage. They capture feature diversity in their test adequacy metric named Input Distribution Coverage (IDC) by computing the interaction between abstract features. While VAEs are effective at extracting related features of the input, they can be less accurate when they encounter inputs that differ from their training set and lead to unrelated connections between features and behaviour of the model.

Attaoui et.al [8] used a pre-trained VGGNet model to extract relevant features of misbehaviour-inducing inputs. Their tool, called SAFE, uses these features to compute root cause clusters and selects unsafe test inputs to improve the DL system through retraining. More specifically, SAFE extracts features of unsafe images using transfer learning and applies a dimensionality reduction method to extract important dimensions. Root cause clusters are determined through clustering, followed by visual inspection conducted by engineers as mandated by functional safety analysis. Engineers supply a new batch of images, known as the improvement set, for model retraining. SAFE chooses the unsafe set, a subset of images from the improvement set, which engineers then label and use for retraining the model to enhance its prediction accuracy. While they automated feature extraction and retraining, their approach still includes manual steps for visual inspection of images for safety analysis and labeling newly generated images.

Neelofar et. al [103] proposed an adequacy metric for black-box testing of autonomous vehicles considering their instance space. An instance space refers to a 2D representation of the test scenarios, defined based on the most effective features of the test scenarios. Their approach requires significant domain knowledge to extract meaningful features from a test scenario.

Our goal is to explore the feature space comprehensively, seeking inputs that demonstrate both high performance and diversity across different regions of this space. By doing so, we aim to reveal various weaknesses inherent to the DL system, enhancing our understanding of its performance intricacies.

Chapter 5

Exploring the Feature Space of Deep Learning-Based Systems through Illumination Search

As discussed in the comprehensive work by Riccio et al. [118] and by Zhang et al. [155], the Software Engineering research community is working hard at adequately testing the functionality of DL systems by proposing a steadily growing number of approaches. Since part of the program logic of these systems is determined by the training data, traditional code coverage metrics are not effective in determining whether their logic has been adequately exercised. Therefore, recent testing solutions aim at maximising ad hoc white-box adequacy metrics, such as neuron [112, 51, 143, 154] or surprise coverage [70], or at exposing misbehaviours [1, 156, 40]. A limitation of these approaches is that their output cannot be directly used to explain the behaviour of the DL system under test, e.g. coverage reports do not provide enough information to understand what input features might have caused misbehaviours. Consequently, the usefulness of these approaches for the developers is strongly limited in practice.

Few approaches [2, 119] use behavioural properties during test generation, but none of them considers the combination of interpretable features of the DL system under test as the target of test generation. This hinders them from exploring the feature space at large and providing a detailed explanation on how the system behaves for qualitatively different inputs.

We introduce DEEPHYPERION-CS which addresses this limitation by offering an interpretable characterisation of DL systems' behaviours. It is the first approach to apply Illumination Search to DL system testing and to provide developers with a feature map, where the automatically generated inputs are positioned based on their characteristics and where the misbehaviours they expose can be interpreted. DEEPHYPERION-CS ś output comprises *feature maps* that indicates the performance of generated inputs in the space of the relevant features. As an example, for testing handwritten digit classifiers,

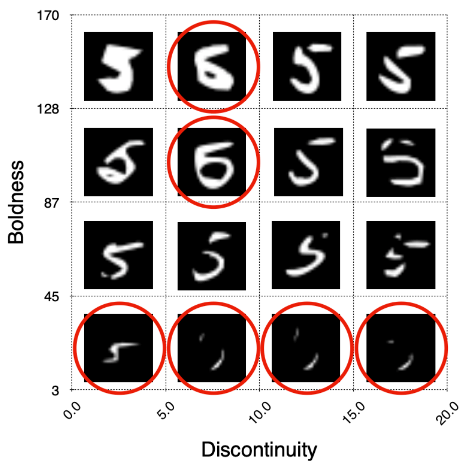


Figure 5.1. Feature map for a handwritten digit classifier. The two axes quantify the discontinuity and boldness of digits. Circled cells highlight misclassified inputs.

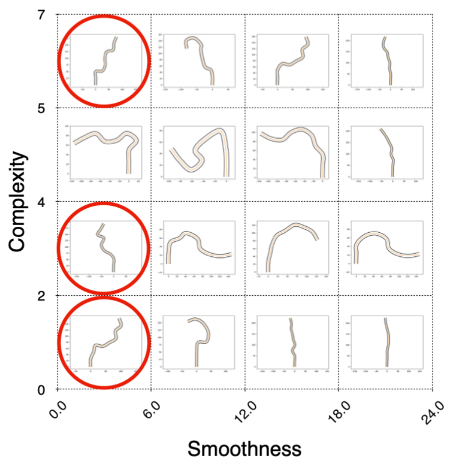


Figure 5.2. Feature map for a lane keeping system. The two axes quantify the complexity and smoothness of virtual roads. Circled cells highlight inputs in which the driving agent went astray.

DEEPHYPERION-CS can use features like the number of disconnected segments within each digit and the boldness of the stroke.

Feature maps are N-dimensional grids, in which each axis corresponds to a considered feature. These maps are discretised so that each of their cells correspond to an interval of features' values. Test inputs are automatically assigned to a map cell, computed by measuring the metric that quantifies each feature. Figure 5.1 illustrates a 2-dimensional feature map for a handwritten digit classifier, where the x-axis corresponds to the Discontinuity feature, while the y-axis corresponds to digits' Boldness. Each feature's value range is discretised into 4 intervals, resulting in a 4×4 map. The resulting feature map highlights that the classifier under test cannot correctly handle thin strokes (i.e., the bottom row of the feature map), as well as bold strokes with moderate discontinuity. Similarly, Figure 5.2 shows that the driving agent under test is in trouble on roads with sharp curves (low values of Smoothness) regardless of their Complexity, where road Complexity is measured as the number of times the road changes direction significantly.

A crucial element of our approach is the choice of the dimensions that define the feature space of interest. In particular, the features should represent meaningful properties of the test scenarios, i.e. discriminative and interpretable properties of the inputs, or behavioural properties manifested by the DL system when exercised by the test inputs. To this aim, we proposed a novel systematic methodology that can be used in conjunction with DEEPHYPERION-CS to define the feature dimensions in a domain of interest, making it possible to generate test cases that illuminate the associated map in such domain. This methodology supports the identification of the features that better characterise the generated inputs and the definition of metrics that quantify the selected features.

We evaluated the proposed technique on both a classification problem (handwritten digit recognition) and a regression problem (steering angle prediction in a self-driving car). Results show that, for both problems, DeepHyperion-CS is effective in generating failure-inducing inputs that are structurally or behaviourally different among them, as they cover different regions of the feature space. We compared DeepHyperion-CS with state-of-the-art test input generators. Our results show that DeepHyperion-CS can explore the feature space at large, whereas existing tools ignore parts of the feature space and expose only misbehaviours that belong to a narrow region of such space.

5.1 Manual Definition of the Feature Map Dimensions

A crucial element of our approach is the choice of the dimensions of variation of the automatically generated test cases. Such dimensions define both the feature space of interest to the user [101] and the search space of DeepHyperion-CS. In the case of DL testing, they should represent meaningful properties of the test scenarios: either discriminative and interpretable *structural features* of the inputs, or *behavioural features* observed as the DL system processes the input and produces its output.

We propose an empirical methodology that can be used to define the feature dimensions in a new domain of interest. Our methodology consists of two macro-steps (see Figure 5.3): (1) *open coding*: select the features that better characterise the generated inputs, and (2) *metric identification*: quantify the selected features. The second step is needed to provide DeepHyperion-CS with quantitative feature values to position the generated tests in the feature map.

5.1.1 Open Coding

The first step entails an *open coding procedure* [128] in which a set of existing inputs is manually analysed by human assessors to select the relevant features in a given domain. Since we are interested in both structural and behavioural features, the information provided to the human assessors is not restricted to the bare inputs (e.g., images or roads): it also includes the output of the DL system when processing the given existing inputs (e.g., the class predicted by an image classifier), as well as any relevant behavioural data (e.g., the trajectory of the car driving on the input road).

The assessors independently tag the inputs assigned to them by either reusing an existing feature label or defining a new one. Each feature label is composed of a feature name, paired with the corresponding feature value, chosen from a rating scale, usually with five levels. For instance, a hypothetical *speed of a self-driving car* label will have

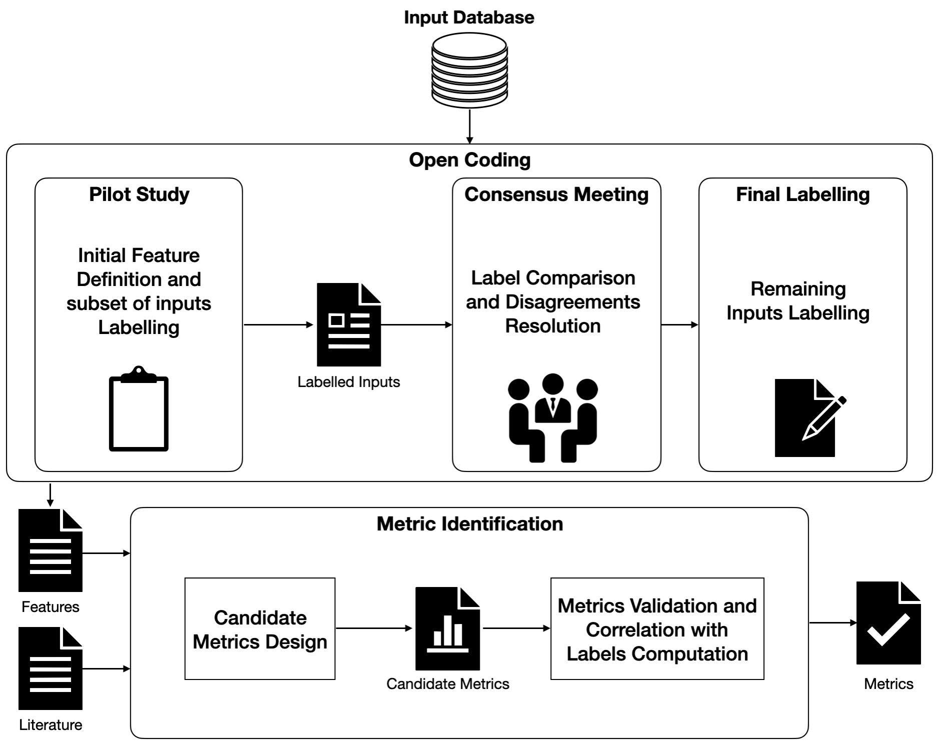
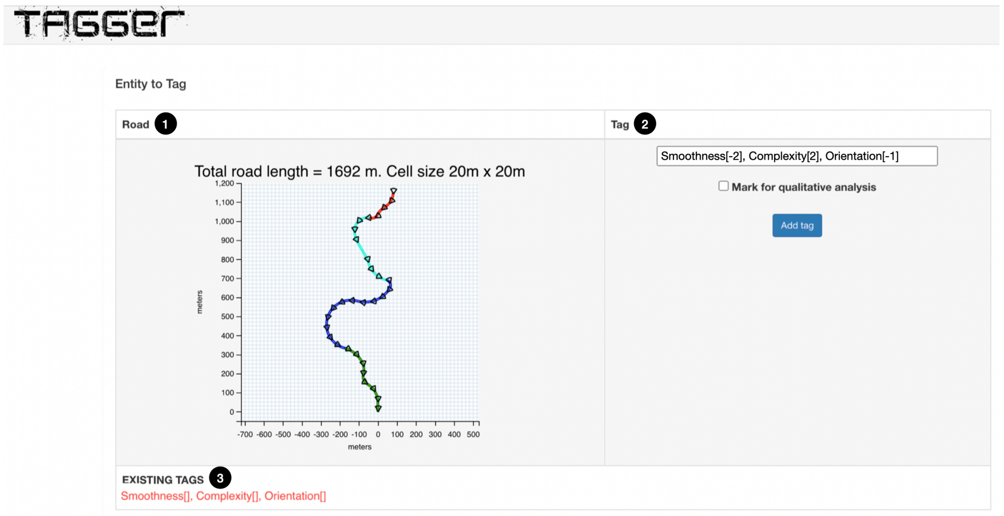

Input Database
Open Coding
Pilot Study
Initial Feature Definition and subset of inputs Labelling
Labelled Inputs
Consensus Meeting
Label Comparison and Disagreements Resolution
Final Labelling
Remaining Inputs Labelling
Features
Literature
Metric Identification
Candidate Metrics Design
Candidate Metrics
Metrics Validation and Correlation with Labels Computation
Metrics

Figure 5.3. Feature Selection Methodology

values that range between -2 and +2, where -2 means “very low”, while +2 means “very high”.

This procedure is supported by a Web application that we developed, which ensures that unlabelled inputs are equally distributed among the assessors and enables assessors to label inputs according to the existing features as well as to define new features. Figure 5.4 shows a snapshot of this Web application for labelling road images. It contains an interactive image of the road with arrows indicating the sample positions of the car in the road ❶ and a text box to assign the label ❷. As shown at the bottom of the figure, the Web application provides the list of already created labels ❸, which can be reused by the assessors. This choice helps them to use consistent naming without introducing substantial bias [60]. In the example in Figure 5.4, the assessor carefully inspected the road shape provided in the left panel ❶ and decided that it has a very sharp angle, a very large number of turns, and covers a moderately small range of directions. Consequently, the assessor filled the text box in the right panel ❷ by assigning a value in the range [-2;+2] to each feature. Notably, the assessor reused the suggested existing tags ❸ as they satisfactorily encoded the identified features. Otherwise, the assessor could have introduced different tags, which would be later proposed to all the assessors in the bottom panel. Based on our experience, this procedure took up to 1 minute for each image.


TAGGER
Entity to Tag
Road 1
Tag 2
Total road length = 1692 m. Cell size 20m x 20m
meters
meters
Mark for qualitative analysis
Add tag
EXISTING TAGS 3

Smoothness[-2], Complexity[2], Orientation[-1]

Smoothness[], Complexity[], Orientation[]

Figure 5.4. A view of the Web application used for labelling the inputs. The (interactive) left panel shows the aerial view of a virtual road divided into road segments. In this panel, little triangles depict the field of view of an hypothetical vehicle driving in the middle of the road, while additional meta-data about the road (e.g., total length) are given for reference. The right panel shows the form used by the labellers to tag the input or define additional tags. The bottom panel shows the set of tags currently defined for virtual roads. Note: a similar page (not shown in the figure) is used to label the images of handwritten digits.

In our methodology, it is strongly advised to run a preliminary pilot study on a subset of inputs to gain confidence in the labelling procedure and, more importantly, agree on the meaning of the features and on the interpretation of the corresponding values. The pilot is concluded with a consensus meeting in which the disagreements are solved either by consensus among the assignees or arbitration by the other assessors. In our experience, a disagreement is worth being discussed in the consensus meeting when the assigned values differ by more than 1 position in the rating scale (e.g., a disagreement between “very low” and “low” speed can be just ignored, while one between “low” and “high” is worth being discussed and solved). It might happen that the assessors realise through the discussion that some important features have been overlooked. Therefore, as part of the consensus meeting, assessors are allowed to agree upon additional features to be considered during the labelling procedure.

Only when a common understanding of the features and of their possible values is reached, we suggest that it is possible to switch from the pilot study to the study mode. In the final study, it is usually enough that each remaining unlabelled input is evaluated by a single assessor. In fact, while during the pilot study the number of inputs being labeled is kept small, in the final labelling phase we typically want to label as many inputs as possible.

Table 5.1. Feature selection and validation: output of the proposed methodology in the two reference application domains

Application Domain	Case Study	Feature	Metric	Agree	Correl	p-value
Digit Recognition	MNIST	Boldness	Lum	100%	0.67	<0.002
		Smoothness	AvgAng	66%	0.05	0.241
		Discontinuity	Mov	100%	0.90	<0.002
		Rotation	Or	100%*	0.43	<0.002
Autonomous Driving	BeamNG	Smoothness	Curv	95.8%	-0.60	<0.002
		Complexity	TurCnt	87.5%	0.63	<0.002
		Orientation	DirCov	89.5%	0.66	<0.002
		Passenger Comfort	StdSA**	–	–	–
		Safety	MLP**	–	–	–

* *Rotation was identified* during *the consensus meeting, after all the assessors agreed upon its meaning (i.e., Agreement=* 100%*).*
***StdSA and MLP were identified in the study about quality of driving metrics for self-driving cars by Jahangirova et al. [63]*

5.1.2 Metric Identification

The second step of our methodology aims to define a set of metrics that can accurately quantify the domain-relevant features. The metrics can be either (1) selected from the most used in the literature or (2) designed ad-hoc to accurately quantify the features identified in the Open Coding step. To select the most accurate metrics for the features that have been identified in the previous step, we compute the Pearson correlation coefficient [73] and the associated p-value, between the manually defined feature values, converted from the rating scale to a numeric scale (e.g., in the range [1:5]), and the values returned by the candidate metrics. The metrics with highest, statistically significant (p-value < 0.05) correlation are chosen to quantify the selected features. In the following, we provide the details about how this methodology was applied to each of our case studies, i.e. digit recognition and autonomous driving.

5.1.3 Dimensions for Digit Recognition

Open Coding

In this phase, I and two colleagues acted as assessors. In the pilot, we randomly selected 30 images from the MNIST database and each assessor was assigned 20 images, such that each input was evaluated by two assessors. The assessors identified the following features, to which they assigned values within a range from -2 to 2:

- **Boldness**, indicates how strong the stroke of the handwriting is. It ranges from very thin (−2) to very thick line (2).

- **Smoothness**, indicates the absence of sharp angles in the digit. It ranges from sharp angles (−2) to smooth angles (2).
- **Discontinuity**, indicates how continuous the stroke of the handwriting is. It ranges from continuous line (−2) to digits made of multiple disconnected segments (2).
- **Rotation** with respect to the vertical axis. It ranges from strongly tilted to the left (−2) to strongly tilted to the right (2).

Examples of images of handwritten digits at various levels of Boldness and Discontinuity can be found in Figure 5.1. The inter-rater agreement during the pilot study, measured as the percentage of inputs that were assigned the same feature value or feature values with a difference of 1, is reported in Table 5.1 under *Agree*. We observed that assessors strongly agreed over Boldness and Discontinuity (i.e., no conflicts have been registered). Noticeably, Table 5.1 does not report any agreement value for Rotation, as the assessors introduced this feature during the consensus meeting, i.e., after the data collection for the pilot study ended. In the final phase, we randomly selected 600 images from MNIST and each of the three assessors labelled 200 images.

Metric Identification

To measure each feature resulting from the labelling procedure, we designed several candidate metrics and applied them to the 630 images labelled by the assessors. Table 5.1 (top) shows the metric with highest correlation for each MNIST feature, together with the corresponding correlation and p-value:

- **Luminosity** (Lum): number of light pixels of the image, i.e., pixels whose value is above 127.
- **Average Angle** (AvgAng) the average angle of the Bezier curves in the SVG representation of the digit.
- **Moves** (Mov): sum of the Euclidean distances between pairs of consecutive sections of the digit. To obtain the sections of a digit, we convert its bitmap to SVG.
- **Orientation** (Or): vertical orientation of the digit, obtained by computing the angular coefficient of the linear regression of the non-black pixels, i.e., pixels with value greater than 0.

As shown in Table 5.1, for Boldness, Discontinuity and Rotation we were able to define metrics that significantly correlate with the human assessment, whereas this was not possible for Smoothness, which turned out to be both difficult to evaluate for humans (see low inter-rater agreement) and difficult to quantify precisely. Hence, this feature was not included among those used by DEEPHYPERION-CS for input generation.

5.1.4 Dimensions for Autonomous Driving

Open Coding

In this phase, I with the colleagues involved in the previous step and an additional colleague acted as assessors. In the pilot, we randomly generated 40 virtual roads according to our model representation. Each assessor was assigned 20 images representing roads, so that each road was evaluated by two assessors. The assessors identified the following features, to which they assigned values within a range from −2 to +2:

- **Smoothness**, indicates how smooth the turns of the road are. It ranges from sharp turns (−2) to gentle turns (+2).
- **Complexity**, indicates how complex the road's shape is. It ranges from almost straight roads (−2) to roads with many turns (+2).
- **Orientation**, indicates how many directions (i.e., N, NE, E, SE, S, SW, W, NW) the road covers. It ranges from straight road which is oriented to one direction only (−2) to road that covers the whole spectrum of directions (+2).

As reported at the bottom of Table 5.1, during the pilot, we observed that assessors generally agreed upon all the features. In the final phase, we randomly generated 400 roads and each assessor tagged 100 of them.

Metric Identification

We designed a set of candidate metrics and applied them to the 440 images labelled by the assessors. We eventually selected the following 3 metrics that best correlate with the corresponding features, as reported in Table 5.1 (bottom):

- **Maximum Curvature (Curv)**, which quantifies road smoothness as the inverse of its turns' radius.
- **Turn Count** (TurCnt): number of turns in the road, where a turn is a change of direction between consecutive road segments by more than 5°.
- **Direction Coverage** (DirCov): number of different angular sectors covered by the directions of the road segment. In particular, we consider 36 sectors, each spanning 10°.

In addition to the features that characterise the structure of the test input, we considered further features to capture the behaviour of the car during the simulation. In particular, we used the following metrics that have been proposed as quality metrics for self-driving cars [63] to measure the *quality of driving*:

- **Standard deviation of the steering angle** (StdSA): standard deviation of the sequence of steering angles collected along the road during self-driving.

- **Mean lateral position of the car** (MLP): mean distance between the center of the car and the center of the driving lane, where the mean is computed across all car positions observed along the road.

5.2 The DeepHyperion-CS Technique

DeepHyperion-CS aims to extensively explore the feature space of a DL system to find inputs with diverse characteristics that induce the system to deviate from the expected behaviour. Given *N* dimensions of variation of interest, which define the feature space (i.e., the feature map to explore), MAP-Elites looks for test inputs that expose misbehaviours in the system under test at each point in the space defined by those dimensions (i.e., the map's cells). Its goal is to fill the feature map with the fittest individuals, i.e., inputs that expose or are close to exposing misbehaviours.

MAP-Elites needs a domain- and problem-specific *fitness function* to measure the degree of misbehaviour exhibited by the system when executed with a given candidate solution as input. For example, when testing Deep Neural Networks (DNNs) that recognise handwritten digits in greyscale images, two dimensions of interest may be the boldness and discontinuity of the handwriting stroke (see section 5.1). In this case, DeepHyperion-CS uses the misclassification distance as fitness function [20, 119, 28] to generate greyscale images containing digits written using strokes with different boldness and discontinuity (see Figure 5.1). The misclassification distance is computed as the difference between the activation value of the neuron associated with the correct label and the highest incorrect activation from the DNN's *softmax* layer output (hence, it becomes negative as a misclassification occurs).

The original MAP-Elites algorithm uses uniform random individual selection to perform the search, i.e., at each iteration, it generates a new input by modifying an individual randomly chosen among the ones already occupying some map cells. The motivation behind this choice is that random selection avoids biasing the search and possibly achieving suboptimal solutions [101]. For instance, selecting the fittest individual at each iteration may drastically reduce the population's diversity and lead to premature convergence of the search.

However, smarter selection mechanisms usually improve search-based algorithms compared to the random baseline, e.g. survival of the fittest [35] and promotion of the most diverse individuals [85, 97]. Therefore, in this approach we integrated a novel selection operator into MAP-Elites, specifically designed to promote individuals that contribute more to the map exploration. The ability of an individual to generate many and diverse new inputs is captured by our novel metric, named *contribution score (CS)* (see section 5.2.5 for a detailed description). Our assumption is that an individual contributes to the search when it generates mutants that occupy previously empty cells or are fitter than existing individuals. If an individual contributes to the search more often, it could be more useful to generate better mutants also in next iterations. On

Algorithm 1: DeepHyperion-CS's Illumination Search

```
Input  : $B$: execution budget
         $featurelist$: list of features
         $seedsize$: seed pool size
         $popsize$: population size
         $rankselectionprob$: rank selection probability
         $rankbias$: rank bias
Output : $M$: feature map
1  map $M \leftarrow$ InitializeMap($featurelist$);
2  seeds $S \leftarrow$ GenerateSeeds($seedsize$);
3  foreach $s \in S$ do
4      Evaluate($s$);
5  end
6  population $P \leftarrow$ InitialisePopulation($S$, $popsize$);
7  foreach $ind \in P$ do
8      $M \leftarrow$ UpdateMap($ind$) ;
9  end
10 while $elapsedBudget < B$ do
11     $ind \leftarrow$ CS-RankSelection(M, $rankselectionprob$, $rankbias$);
12     $ind_\mu \leftarrow$ Mutate($ind$);
13     Evaluate($ind_\mu$);
14     $M \leftarrow$ UpdateMap($ind_\mu$);
15     $ind \leftarrow$ UpdateCS($M$);
16 end
17 return ($M$)
```

the other hand, if an individual does not contribute to the search for several iterations, it is unlikely that it will generate better mutants later; in this case, we progressively reduce that individual's CS score in order to give it a lower priority during the selection. Consequently, selecting individuals with higher CS can lead to fill more cells of the feature map, possibly in fewer iterations, than uniform random selection. Moreover, exploring more feature combinations may also lead to exposing more misbehaviours.

Algorithm 1 outlines the high-level steps of the Illumination Search approach implemented by DeepHyperion-CS. The algorithm starts by filling an empty N-dimensional and discretised feature map M (line 1) with an initial population P to be evolved (line 6), where N is the number of features in the *featurelist* provided as input. The initial population is drawn from a pool S of valid candidate inputs, called *seeds*, that are evaluated using the EVALUATE function which computes the fitness function (i.e. closeness to misbehaviour) and the features' values of the considered individual (lines 3–5). The cost of input evaluation is domain-dependent and spans from a simple model prediction, e.g., for handwritten digits, to performing expensive simulations, e.g., for lane keeping. On the basis of the computed features, the candidate inputs are placed into M following the update map rule (lines 7–9), i.e., each map cell can be occupied only by the fittest individual with feature values corresponding to that cell. After creating P, the algorithm performs the main evolutionary loop (lines 10-15) until a termination condition on the execution budget is met. At each loop iteration, an individual *ind* is chosen from the current map using the CS-based rank selection operator (line 11). The selected individual is mutated to generate a new input ind_{μ} (line 12) and, then, its features and fitness are evaluated (line 13). The map is updated with ind_{μ} (line 14): if it has a higher fitness value than the individual in the map cell it occupies, it replaces the existing entry in the map (this is done also if the map entry is currently empty). Finally, the CS of the parent individual *ind* is updated according to whether its mutant ind_{μ} contributed to the exploration or not (line 15). In the next sections, we detail the key aspects of DeepHyperion-CS and describe how we applied it to the chosen application domains.

5.2.1 Input Representation

DeepHyperion-CS is a model-based test input generation technique [146]: it generates complex inputs (e.g., greyscale images) by manipulating a *model* of the input, instead of directly modifying the raw input data (e.g., pixels). Consequently, DeepHyperion-CS requires a generative model of the input data processed in the application domain. Generative input models are largely domain-specific and are commonly employed in several domains, including safety-critical ones [79].

A possible alternative to model-based manipulation could be to directly modify the raw input data (e.g., pixels for MNIST) as done in traditional adversarial Machine Learning (ML). Adversarial ML techniques focus on applying the minimal changes that can trigger a misbehaviour and are guaranteed to achieve this goal [16]. However, they are not focused on generating inputs with different structural features and, thus, covering the feature map. Another alternative for input generation are generative ML approaches that approximate the input distribution, such as Variational Auto-Encoders (VAEs) [67] and Generative Adversarial Networks (GANs) [28]. VAEs and GANs are very useful when a model of the inputs is not available, e.g., real-world images from ImageNet. However, generative ML based approaches rely on the quality of both a representative training set and trained generative ML models, which might be hard to achieve for complex problems.

We evaluated DeepHyperion-CS on two reference problems, handwritten digit recognition in the image classification domain and lane-keeping in the automotive domain.

For the handwritten digits recognition problem, we refer to the image format adopted by the MNIST database [82] that consists of 70 000 greyscale 28×28 images of handwritten digits. DeepHyperion-CS abstracts each digit as a sequence of (start, end, and control) points that define Bézier segments by utilising the Potrace algorithm [129] and

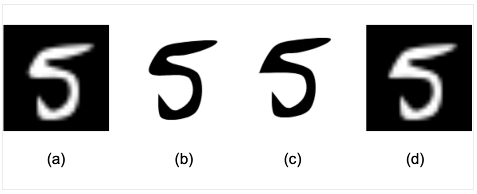


Figure 5.5. Digit input representation and mutation. (a) original input; (b) original SVG model after vectorization; (c) SVG model mutated by moving a control point; (d) mutated input.

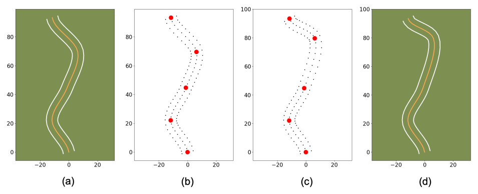


Figure 5.6. Road input representation and mutation. (a) original input; (b) original model; (c) model mutated by moving a control point; (d) mutated input.

stores them as Scalable Vector Graphics (SVG)[1] files, as shown in Figure 5.5. In particular, Potrace performs a sequence of operations, including binarisation, despeckling and smoothing, which draw a smooth contour made of Bezier segments around the considered image. We used Potrace since it represents the state of the art in vector model extraction from images, (see, e.g., the Inkscape tool[2]), and can be easily integrated into Python code via the `pypotrace`[3] API.

For the lane-keeping problem, we refer to the simulated driving scenarios defined by state-of-art approaches for testing lane-keeping systems [119, 40, 110] that consist of flat, two-lane, two-way, asphalt roads surrounded by green grass on which the ego-car (i.e., the vehicle under test) has to drive on the right lane. The environment is set to a clear day without fog. DeepHyperion-CS abstracts roads as sequences of control points in a bi-dimensional space. DeepHyperion-CS interpolates the control points using Catmull-Rom cubic splines [21] to transform them into virtual roads to be rendered in the simulator. Figure 5.6 shows the control points of the centre line spline as larger red dots and the interpolated points that define the road as smaller grey dots.

5.2.2 Fitness Function

Intuitively, a suitable fitness function for testing DL systems should quantify how close the DL system is to exhibit a misbehaviour [119, 40, 56]. In the following, we describe the two fitness functions we designed to address the handwritten digit recognition and lane-keeping problems, respectively.

For the handwritten digit recognition problem, we rely on the fact that the DNN under test recognises the digits in the input image by selecting the class with the highest activation level in its softmax output layer [42]. Therefore, by computing the difference between the activation level of the neuron associated with the correct class and the maximum activation level associated with the other, incorrect classes, we can effectively

[1] `https://www.w3.org/Graphics/SVG/`
[2] `https://wiki.inkscape.org/wiki/Potrace`
[3] `https://github.com/flupke/pypotrace`

measure whether the prediction was correct (positive fitness value) or wrong (negative fitness value). More importantly, using this fitness function, DeepHyperion-CS can measure how close an input is to cause a misbehaviour and can expose misbehaviours by minimising the fitness value.

For the lane-keeping problem, we adopt a fitness function that scores higher tests causing the ego-car to drive closer to, or even across, the lane's margins. Specifically, DeepHyperion-CS calculates the fitness of a test as $\min(w/2-d)$, where w is the width of the lane the ego-car travels on, and d is the distance of the ego-car from the lane centre. The position of the car is approximated by its centre of mass. The fitness function returns its maximum value $w/2$ when the car is at the lane centre. DeepHyperion-CS aims to minimise this fitness function causing the ego-car to drive over the lane's margins (negative fitness values).

Since it is more important to find all unique misbehaviours that happen in different conditions, rather than finding test inputs that cause a "large amount" of misbehaviour (large negative value of the fitness function), the fitness function is capped to a small negative value (i.e., −0.1) independently of the misbehaviour, thus avoiding that DeepHyperion-CS ends up replacing individuals that caused already discovered misbehaviours with individuals causing more extreme misbehaviours that happen in similar or the same conditions. This strategy has the advantage of making it hard for the fittest individuals to dominate the selection, which might lead to premature convergence [6].

5.2.3 Feature Map

A feature map represents the feature space defined by N dimensions of variation (i.e., the features) that are relevant for characterising the tests generated by DeepHyperion-CS.

DeepHyperion-CS characterises each individual by placing it into the feature map M using the following mapping function:

$$x_i = \lfloor \alpha_i \cdot ind.f_i \rfloor \tag{5.1}$$

where $ind.f_i, \forall i \in [1:N]$ refers to an individual's feature values. According to Equation 5.1, DeepHyperion-CS computes the i-th index of a cell (i.e., the integer x_i that defines its coordinate along the i-th dimension) by scaling the feature value $ind.f_i$ using the scaling factors α_i.

It should be noticed that if a feature f_i can have negative values, the resulting index x_i becomes also negative. Correspondingly, the feature map will span between negative and positive integers (one way to achieve this in the implementation is to use the index x_i as a hash key and to display a grid spanning along all keys). DeepHyperion-CS uses α to control the map's granularity based on the expected range of each feature and to transform continuous values into the map coordinate system, which is based on integers. The granularity of the feature map (i.e., the number of cells along each dimension) is decided by the user of our approach when setting the scaling factor α_i. Specifically, α_i can be empirically computed as the ratio between the desired granularity, i.e., the desired number of cells, and the expected range of the corresponding feature f_i. The choice of the map granularity affects its discriminative power, since a too low granularity might be insufficient to characterize the misbehaviours and to distinguish them from correct behaviours. However, in our experience any reasonably high choice (as a rule of thumb, more than 25 cells) is enough to ensure good discrimination.

For instance, if the i-th feature's values are expected to range between 0.0 and 2.0, and the desired granularity is 100, a suitable value of α_i would be 50. The α values remain constant during the search, while the size of the map M dynamically increases as DeepHyperion-CS generates individuals with feature values outside the current map boundaries. Initially, M contains no cells; then, as soon as DeepHyperion-CS generates new tests with features that map to indexes outside the current range of values, it grows M to accommodate the newly discovered individuals and adjusts the range of values along the extended dimensions. For instance, if the first mapped individual in a hypothetical bi-dimensional map has indexes $(x_1, x_2) = (2, 3)$, the initial empty map M would be updated to have one cell in each direction at position $(2, 3)$ and ranges $([2:2], [3:3])$. If later DeepHyperion-CS maps another individual to $(x_1, x_2) = (5, 1)$, M grows along its first dimension to the range $[2:5]$ and to $[1:3]$ along the second dimension. At this point, the feature map contains 12 cells, 2 of which are filled, i.e., they contain an individual.

Since the size of the final dynamically discovered map may be different between various runs of the algorithm, which may hinder visual inspection of the results, DeepHyperion-CS allows testers to define the granularity of the final map produced by the algorithm. It produces the final map by rescaling the search results as follows:

$$x_i' = \left\lfloor \mathrm{GS}_i \cdot \frac{ind.f_i - min_i}{max_i - min_i} \right\rfloor \tag{5.2}$$

where GS_i is the desired grid size of the final map, and min_i and max_i are the minimum and maximum values empirically observed for the i-th feature. Rescaling the feature maps eases the comparison of maps produced by DeepHyperion-CS and other test generators across various runs, when results have different ranges. Therefore, we rely on map rescaling in the experimental evaluation.

In particular, we rescale the maps so that they have the same size and features' ranges across all runs of all test generators, hence avoiding any misalignment between maps.

The generation of multiple inputs that belong to the same cell, e.g., through multiple DeepHyperion-CS's runs, allows the identification of feature map regions where the probability of observing misbehaviours is higher. In fact, a combination of feature values that often corresponds to a misbehaviour may suggest that such feature values are very likely to induce a misbehaviour. In this way, DeepHyperion-CS provides developers with

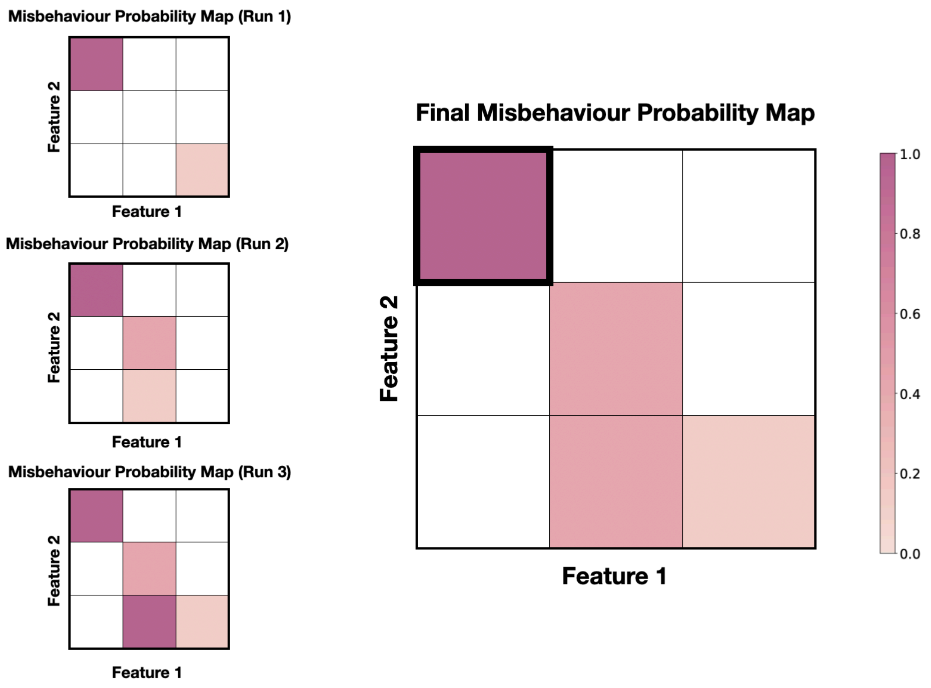


Figure 5.7. Misbehaviour probability maps: darker cells correspond to feature combination values that are more likely to induce misbehaviours, dark borders highlight cells with high confidence of producing a misbehaviour.

a powerful tool to understand the causes of misbehaviours. Therefore, we synthesise the information collected by DeepHyperion-CS across multiple runs (i.e., all the generated inputs and the corresponding outputs) in *misbehaviour probability maps*, that report the Average Misbehaviour Probability (AMP) associated with each cell. We compute these maps by (1) measuring, within each cell, the ratio of the number of misbehaviour-inducing inputs to the total number of inputs generated by DeepHyperion-CS during each run, and then (2) averaging the resulting values per cell across all the tool's runs. Since DeepHyperion-CS may generate only a small number of inputs in some cells, the corresponding AMP values might be affected by a large error. Hence, we also compute the confidence interval of AMP. In particular, we use Wilson's confidence interval estimator for binomial random variables, which indicates whether the misbehaviour probability estimated for a particular combination of feature values has a low or high error range. We consider a combination of feature values to produce misbehaviours with high confidence if its AMP value is greater than 0.8 and the lower bound of its confidence interval is above 0.65. As shown in Figure 5.7, misbehaviour probability maps contain blank cells corresponding to feature combination values that have never been observed, while the other cells are shaded proportionally to their AMP values. Combinations producing misbehaviours with high confidence have thick borders.

5.2.4 Initial Population

DeepHyperion-CS generates an initial population of size *popsize* by choosing inputs from a larger pool of seeds of size *seedsize*, consisting of valid inputs for the system under test. For the handwritten digit recognition problem, these seeds are existing images randomly drawn from the MNIST database and converted to SVG, while for the lane-keeping problem, seeds are valid roads that are generated randomly.

Generation of DeepHyperion-CS's initial population aims to create a set of diverse individuals from the feature space. Therefore, after evaluating the seed fitness and computing the seed position on the map based on the feature values, DeepHyperion-CS greedily selects individuals from the seed pool so as to maximise their Manhattan distance (sum of the absolute differences of the map coordinates) [76].

5.2.5 Contribution Score-Based Rank Selection

As shown in Algorithm 1, each DeepHyperion-CS's evolutionary iteration starts by selecting an existing individual to be mutated from the non-empty cells of the feature map. To select such individual, DeepHyperion-CS can use either a random strategy (RandomSelection) or our novel strategy based on the contribution score (CS-RankSelection). The rank selection probability parameter *rankselectionprob* controls the frequency of usage of each selection strategy. Specifically, increasing values of *rankselectionprob* result in adopting CS-RankSelection more frequently than RandomSelection.

RandomSelection is a standard selection strategy, in which an individual is uniformly sampled among the ones already placed in the feature map.

CS-RankSelection implements a rank selection scheme which selects individuals with a probability proportional to their rank, such that high-ranked individuals are selected with higher probability than low-ranked ones.

Individuals are ranked by CS-RankSelection according to their *Contribution Score (CS)*, which represents the individual's contribution to exploration. An individual has contributed to exploration if it has generated mutants which filled previously empty cells or which replaced existing individuals with better ones. In detail, CS is computed as follows:

$$CS(x) = \begin{cases} \frac{CC(x)}{SC(x)} & \text{if } SC(x) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

where $CC(x)$, the *Contribution Count*, indicates the number of times mutants of individual x have been successfully placed in the map, while $SC(x)$, the *Selection Count*, indicates the number of times the individual has been selected during the search. According to its definition, CS is always bounded between $(0, 1]$. Each individual's CS is initially set to 1, which means that 1. all the individuals are selected with equal probability before collecting any observation (unbiased initial selection); and 2. individuals who have never been selected are more likely to be selected than others (promoted exploration).

During the search, CS values are updated to reflect each individual's actual contribution. For instance, if an individual x_1 with $CC(x_1) = 1$ and $SC(x_1) = 1$ is selected but its mutant is not placed on the map (i.e., the existing individual already placed into its same cell has higher fitness), $SC(x_1)$ increments but $CC(x_1)$ remains the same. As a result, $CS(x_1)$ drops from a solid 1.0 to a less considerable value of 0.5, halving the chances to select x_1 in the following iterations. Interestingly, the contribution score of an individual does not always monotonically decrease during the search since every time $SC(x)$ increases, the corresponding value of $CC(x)$ may or may not increase. The initial value of $CS(x)$ is 1.0 to promote individuals that have not been yet selected. Regime values for $SC(x)$ are usually big, making the difference between contiguous values of $CS(x)$ small, i.e., CS is overall smoothly changing. However, there is an initial, transient phase where by design $CS(x)$ is less smooth, e.g. jumping from 1.0 to CS = 0.0 (non contributing individual) and then to CS = 0.5 (contributing individual), to quickly lower the rank of individuals that proved not to contribute to the search.

To perform rank selection of individuals, we use the linear ranking function proposed by Whitley [151]. This approach to rank individuals is widely used in search-based software testing since it addresses the problem of maintaining a constant selective pressure of genetic algorithms throughout the search [98, 34, 41, 9, 54, 55]. In accordance with this technique, DEEPHYPERION-CS sorts the individuals in ascending order based on their CS value. Then, it selects from the ordered list the individual corresponding to the index computed with the following formula:

$$index = size(Individuals) \times \frac{\left(\sqrt{rankbias^2 - 4 \times (rankbias - 1) \times random(0,1)}\right)}{2.0 \times (rankbias - 1)} \tag{5.4}$$

where function *size* returns the length of a list and the function $random(0,1)$ returns a random floating point number between 0 and 1. The *rankbias* parameter ranges between 1.0 and 2.0 and influences the algorithm's behaviour by biasing the selection towards individuals with higher ranks (i.e., *rankbias* values close to 2.0) or lower ranks (i.e., *rankbias* values close to 1.0) [151]. In the former case, the selection operator does not always select the best individual, which helps to avoid local optima.

In summary, by adopting our rank selection operator based on contribution score, DEEPHYPERION-CS can bias the search towards solutions with high contribution scores, hence exploring the feature space at large, "illuminating" the search space as much as possible. Additionally, by exposing parameters such as *rankselectionprob* and *rankbias*, DEEPHYPERION-CS enables testers to control the selection pressure and the level of bias imposed by the rank selection on the overall search process.

5.2.6 Model-Based Mutation Operators

After selecting an individual, DeepHyperion-CS mutates it to generate a new input. Since DeepHyperion-CS is a model-based test generator, its mutation operators manipulate an input model rather than the input itself. DeepHyperion-CS uses mutation operators that apply small perturbations to the input models within a customisable range.

For the handwritten digit recognition problem, DeepHyperion-CS manipulates the SVG image model's points to mutate the corresponding digit shape while preserving realism [119]. Then it applies a rasterisation operation to obtain the input in the MNIST database format[4] (see Figure 5.5). For the lane-keeping problem, DeepHyperion-CS mutates the road geometry by applying a displacement to the coordinates of the model's control points (see Figure 5.6).

Despite the small perturbations applied by DeepHyperion-CS, the generated mutants may not be different from their parents or even valid once concretised into actual test inputs. Therefore, DeepHyperion-CS verifies that the mutants are different from their parents and comply with the constraints of the input domain before evaluating them. DeepHyperion-CS keeps mutating the same parent individual until a valid mutant is found.

For the handwritten digit recognition problem, DeepHyperion-CS (i) computes the Euclidean distance between the mutant and its parent, which must be greater than 0; (ii) computes the Euclidean distance between the mutant and the starting seed, which must be greater than 0 and lower than 2.

For the lane-keeping problem, it checks that mutated roads (i) do not have control nodes identical to the parent's control nodes; (ii) are entirely contained within a squared bounding box of fixed size (i.e., the driving simulator's map boundaries); and (iii) do not self-intersect.

5.3 Experimental Evaluation

5.3.1 Research Questions

The goal of our evaluation is to understand whether coupling feature maps and automated test generation is an effective technique for DL testing, which (1) can thoroughly stress the DL system under different conditions, and (2) can provide information useful to characterise problems in DL systems. Therefore, we seek to answer the following research questions:

RQ1 (Failure Diversity): *How effective is* DeepHyperion-CS *in generating test inputs that expose diverse failures?*

Generating tests that trigger failures is more useful when these failures are diverse. Whereas, a test generator that repeatedly exposes the same problem is not desirable, as

[4]DeepHyperion-CS utilises the open-source graphic libraries LibRsvg and Cairo for rasterising SVG images to the MNIST format.

it wastes computational resources.

Metrics: To assess how many different failures are triggered during a run, we measure the number of *Mapped Misbehaviours (MM)*, i.e. how many cells of the feature map M contain at least one failure-inducing input.

To measure the diversity of the misbehaviour-inducing inputs, we compute the *Misbehaviour Sparseness*. In particular, we compute the average Manhattan distance between cells containing misbehaviours and the average maximum Manhattan distance between cells containing misbehaviours. We consider two slightly different sparseness metrics to take into account outliers and denser map regions:

$$\text{Misbehaviour Sparseness (Avg. Max)} = \frac{\sum_{i \in MM} \max_{j \in MM} \text{dist}(i,j)}{|MM|} \tag{5.5}$$

$$\text{Misbehaviour Sparseness (Avg.)} = \frac{\sum_{i,j \in MM, i \neq j} \text{dist}(i,j)}{|MM|(|MM|-1)} \tag{5.6}$$

RQ2 (Search Exploration): *How extensively does* DEEPHYPERION-CS *explore the feature space?*

Effective test generation should exercise different behaviours of the systems under test. This can be achieved by exploring the feature space extensively, at large.

Metrics: We measure the thoroughness of exploration by counting the *Filled Cells (FC)* in the map, i.e., the cells of the feature map that contain at least one input. We quantify how broadly those filled cells spread over the feature space by measuring their sparseness, i.e., the *Coverage Sparseness*. Similarly to Misbehaviour Sparseness, we consider the following two sparseness metrics:

$$\text{Coverage Sparseness (Avg. Max)} = \frac{\sum_{i \in FC} \max_{j \in FC} \text{dist}(i,j)}{|FC|} \tag{5.7}$$

$$\text{Coverage Sparseness (Avg.)} = \frac{\sum_{i,j \in FC, i \neq j} \text{dist}(i,j)}{|FC|(|FC|-1)} \tag{5.8}$$

RQ3 (Efficiency): *How efficient is* DEEPHYPERION-CS *in exploring the feature space and generating test inputs that expose diverse misbehaviours?*

Testing DL systems can be costly, especially when it is conducted at the system level, as happens, e.g., with simulation-based testing of self-driving cars. Therefore, we evaluate how quickly test generators fulfill the testing objectives of triggering misbehaviours and extensively exploring the feature space.

Metrics: We assess test generation efficiency by measuring the *Area Under the Curve (AUC)* of Mapped Misbehaviours and Filled Cells. AUC is a standard performance metric, and higher values of AUC indicate more efficient test generators.

RQ4 (Training Data Expansion): *Can* DEEPHYPERION-CS *be used to expand the training data? Can it find misbehaving inputs also in cells that were already occupied by non-misbehaving training data?*

The knowledge acquired by a DL system is limited by the diversity of the data that have been used to train it. We evaluate how DeepHyperion-CS can expand such knowledge beyond the training set, by identifying feature combinations that are not covered by the existing training set. New input data generated for such initially uncovered feature map cells can expand the training set and increase the generalisation capability of the system.

Such initially uncovered cells are particularly interesting when they contain an input triggering a misbehaviour. On the other hand, DeepHyperion-CS's fitness-guided local competition can also generate misbehaviour-inducing inputs for feature combinations that were not associated to any misbehaviour in the training set. Therefore, we also evaluate how many unknown misbehaviours are triggered by DeepHyperion-CS in cells that are either covered or uncovered by the training set.

Metrics: We answer this question by comparing the feature maps produced from the training set with those generated by DeepHyperion-CS. We measure the size of *Filled Cell Expansion* (FCE), computed as the cells filled by DeepHyperion-CS that were uncovered in the training set:

$$\text{FCE} = |FC_{DH} \setminus FC_{ts}| \tag{5.9}$$

where FC_{DH} is the set of cells filled by DeepHyperion-CS, whereas FC_{ts} is the set of cells filled with training set data. To measure how DeepHyperion-CS is able to generate new misbehaviours, we define *Mapped Misbehaviour Expansion* (MME) as:

$$MME = |MM_{DH} \setminus MM_{ts}| \tag{5.10}$$

where MM_{DH} is the set of cells containing misbehaviours in maps generated by DeepHyperion-CS, whereas MM_{ts} is the set of cells containing misbehaviours in the training set. MME consists of two distinct sets: (1) the misbehaviours generated by DeepHyperion-CS in cells not covered by the training set (MME_{uncov}); and, (2) the misbehaviours generated by DeepHyperion-CS in cells that are covered by training set data, but only by correctly behaving inputs (MME_{cov}). We define these sets as follows:

$$MME_{uncov} = MME \setminus FC_{ts} \tag{5.11}$$

$$MME_{cov} = MME \cap FC_{ts} \tag{5.12}$$

5.3.2 Experimental Procedure

We addressed our research questions by running DeepHyperion-CS and other state-of-the-art test input generators against the two considered test subjects. At the end of the runs, we used the results generated by each tool to compute the corresponding feature maps. To ensure a fair comparison, all the maps were generated with the same number

Table 5.2. DeepHyperion-CS Configurations

Parameter	**Test Subject**	
	MNIST	BeamNG
seed pool size	900	80
population size	800	48
time budget (s)	3600	36000
mutation range lower bound	0.01	1
mutation range upper bound	0.6	6
ranked selection probability	0.5	0.5
rank bias	1.5	1.5
feature combinations	(Mov, Or)	(MLP, StdSA)
	(Or, Lum)	(MLP, TurnCnt)
	(Lum, Mov)	(StdSA, Curv)

of cells for each feature, i.e. up to 25 cells. The extreme values defining the range for each feature are the ones observed across the runs of all the tools. Since the final maps produced by different tools may have different ranges and higher number of cells, we rescaled them by using the formula described in Equation 5.2.

Since the test subjects are fundamentally different, we adopted two separate configurations for testing them (see Table 5.2). We empirically obtained those configurations after observing DEEPHYPERION-CS's behaviour in few preliminary runs. Specifically, DEEPHYPERION-CS obtained the seeds for MNIST by randomly selecting 900 inputs from the official MNIST test set, all belonging to the same class (i.e., digit "5") and then selecting the 800 most diverse inputs as initial population. The seeds for BEAMNG were 80 valid roads randomly generated by DEEPHYPERION-CS. Each seed was defined by 10 control points in which the initial point was always at a fixed position, whereas the remaining points were placed at a random position 25 meters away from the previous one and deviating from the previous segment by an angle randomly chosen within a predefined range. DEEPHYPERION-CS then selects the 48 most diverse roads as initial population.

As regards the selected feature dimensions, we used the features identified by applying our methodology (see Section 5.1). We considered only pairwise combinations of features to ease visualisation and discussion of the results, although DEEPHYPERION-CS can work also with higher-dimensional maps. For MNIST, we considered all the pairs obtained by combining Boldness (Lum), Discontinuity (Mov), and Rotation (Or), as these are the most significant features we found in our feature selection study (see Table 5.1). For BEAMNG, we considered three out of ten possible pairs of features because executing driving simulations becomes soon prohibitively expensive and running experiments that cover all the possible combinations would take excessive computation time. Nevertheless, we believe that the results we achieved are representative as we cover one combination of two behavioural features and two combinations of a structural and a behavioural feature.

To contextualise the results achieved by DeepHyperion-CS, we compare it against the DeepHyperion (the tool without CS guidance mechanism) and other state-of-the-art testing tools for DL systems. We configured those approaches according to the configurations that achieved the best performance in their papers.

Specifically, we compared DeepHyperion-CS against:

DeepHyperion This tool is the same as DeepHyperion-CS, except that it adopts the original MAP-Elites algorithm, i.e. it uses random uniform selection to perform the evolutionary search.

DLFuzz [51] This tool generates adversarial inputs for image classifiers, such as MNIST, by applying perturbations to the pixels of existing images. It is mainly used for testing the robustness of DL systems. However, since it can only manipulate individual images, we could not apply DLFuzz for testing BeamNG at the system level;

DeepJanus [119] This tool generates test inputs at the frontier of behaviours of DL systems, i.e., pairs of similar inputs that trigger different system behaviours, by using a multi-objective search algorithm. DeepJanus shares with DeepHyperion-CS the same model-based input representation; hence, we could apply it to both MNIST and BeamNG;

AsFault [40] This tool generates safety-critical virtual roads for testing lane-keeping systems utilising a single-objective genetic algorithm. Therefore, we could apply it for testing only BeamNG.

To enable a fair comparison with AsFault, we replaced its original failure identification mechanism with the one employed by the other tools (i.e., DeepHyperion-CS and DeepJanus). Additionally, since AsFault generates longer roads than the other tools, we split each road it generates into multiple segments when placing the inputs on feature maps, making it possible to directly compare its output with the other tools. However, in this way a single AsFault input may contribute to coverage of more than one cell in the feature map. While this might introduce an unfair advantage for AsFault, it is balanced by the increased time it takes to simulate longer roads. Therefore, given the same simulation budget, AsFault generates fewer inputs than the other tools, but each such input covers multiple cells.

As the purpose of the considered baseline tools is finding misbehaviour-inducing inputs or frontier inputs, not illuminating the feature space, in our experiments we consider the inputs generated (and possibly discarded) during the input generation process, in addition to the inputs reported as final outputs of the tools. In particular, we report separately the feature maps obtained from the final results of each tool *(black box results)* and the feature maps which contain all the inputs produced during a run of each tool *(white box results)*. Noticeably, for DEEPHYPERION-CS and its variation the white box and black box maps coincide by construction, as all generated inputs are used during the search in the local competition within each cell. For ASFAULT, white box results also coincide with black box results, as this tool returns all the inputs generated during the search.

We followed the guidelines by Arcuri and Briand [5] for comparing the considered randomised algorithms: we ran each tool multiple times and assessed the statistical significance of our conclusions by performing the Mann-Whitney U-test and measuring the effect size using the Vargha-Delaney's $\hat{A}_{12}$ statistic. To enable a fair comparison, we ran the tools in isolation on the same computing nodes and used the same generation budget: 1 hour for MNIST and 10 hours of simulation time for BEAMNG. We considered simulated time rather than real time for BEAMNG since the former is usually the bottleneck in simulation-based testing [2].

The reason for this remarkable difference between the generation budgets is that testing MNIST consists of feeding it small images and getting the corresponding predictions, an operation that takes milliseconds, while testing BEAMNG requires the execution of real-time driving simulations that take minutes to complete. Correspondingly, we were able to repeat the MNIST experiments 30 times, whereas we repeated the BEAMNG experiments between 10 and 20 times, adopting the following stopping condition (after 10 runs): no further repetition is conducted when the statistical test used to compare the considered tools reaches statistical significance (p-value < 0.05) or when no statistical significance is reached, but there is sufficient statistical power (statistical power $\beta > 0.8$). In no case the number of repetitions exceeds the upper bound, 20.

5.3.3 Results

RQ1: Failure Diversity

In this RQ, we investigate how many diverse inputs the test subjects failed to handle correctly (i.e., *Mapped Misbehaviours*) and how much they differ between them (i.e., *Misbehaviours Sparseness*).

Figure 5.8 reports the results achieved by the considered tools on MNIST as box plots grouped by feature combination.

As DEEPJANUS and DLFUZZ output only a subset of the generated inputs at the end of each run, we considered separately the misbehaviours reported at the end of the run (i.e., black box analysis) and the (possibly bigger) set of all the misbehaviours triggered during the same run (i.e., white box analysis).

Correspondingly, the boxes labelled as DEEPJANUS-WB report more misbehaviours than DEEPJANUS-BB. DLFUZZ's final results (DLFUZZ-BB) show the same values of the boxes obtained from all its generated inputs (DLFUZZ-WB) because this tool returns to

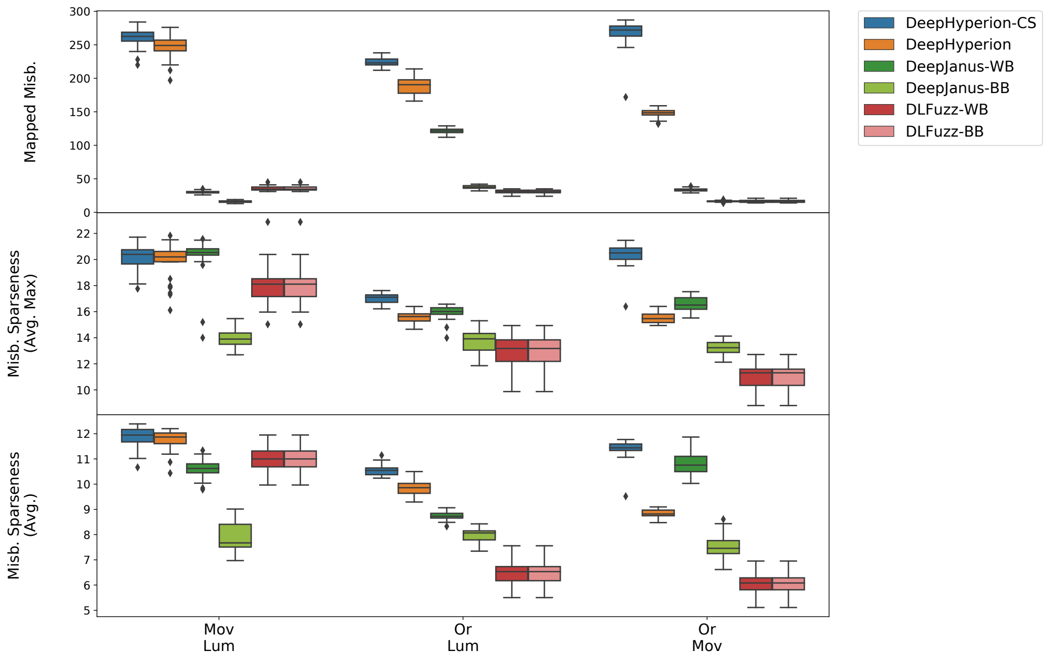


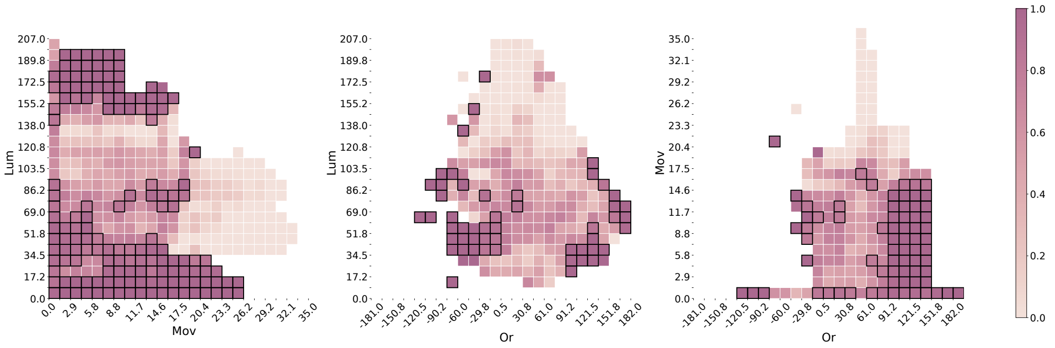
Figure 5.8. RQ1: Mapped misbehaviours found on MNIST by the considered tools (top) and their sparseness (middle and bottom).

the user all the misbehaviours it triggers during a run.

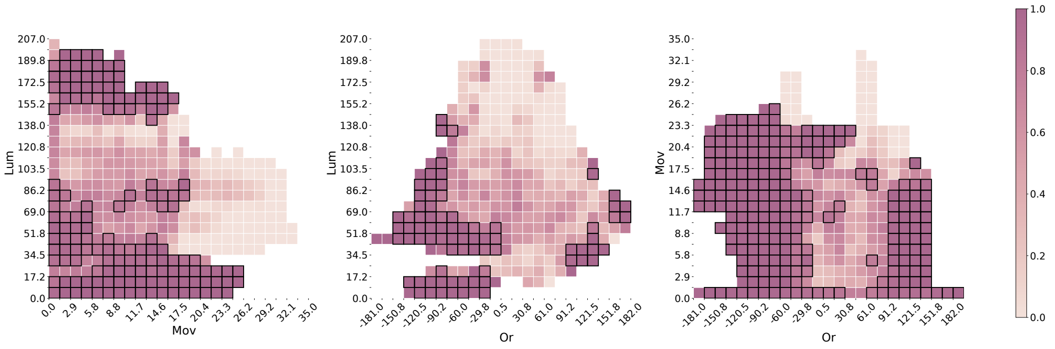
Figure 5.8 (top) shows that DEEPHYPERION-CS found more than 200 diverse misbehaviours for each feature combination. The illumination search based tools, i.e. DEEPHYPERION and DEEPHYPERION-CS, always found a significantly larger number of mapped misbehaviours than the other tools (p-value < 0.05 and large effect size). DEEPHYPERION-CS significantly improves DEEPHYPERION's effectiveness in triggering diverse misbehaviours (p-value < 0.05 and large effect size). In particular, DEEPHYPERION-CS produced a neatly higher number of mapped misbehaviours than DEEPHYPERION for the `Or-Mov` feature combination, with over 120 more misbehaviours.

Both misbehaviour sparseness metrics reported in Figure 5.8 show that DEEPHYPERION-CS produced comparably sparse or sparser misbehaviours than the other tools. In the majority of feature combinations (i.e. `Or-Lum` and `Or-Mov`), it produced significantly sparser misbehaviours than the other tools (p-values < 0.05 and large effect size). As for `Mov-Lum`, DEEPHYPERION-CS performed as good as DEEPHYPERION and DEEPJANUS-BB (p-values > 0.05) and significantly better than the other tools (p-values < 0.05 and large effect size) in terms of Misbehaviour Sparseness (Avg. Max). Moreover, DEEPHYPERION-CS is significantly better than DEEPJANUS-BB for `Mov-Lum` in terms of Misbehaviour Sparseness (Avg.) (p-values < 0.05 and large effect size).

This result was achieved despite DEEPJANUS explicitly rewards the generated inputs' diversity, having a fitness function that promotes the euclidean distance among solutions.



(a) DeepHyperion



(b) DeepHyperion-CS

Figure 5.9. Misbehaviour probability maps generated by DeepHyperion (a) and DeepHyperion-CS (b) for MNIST

We further analyse the relationship between the results achieved by DeepHyperion-CS and DeepHyperion by comparing their misbehaviour probability maps (see Figure 5.9 and Figure 5.11).

The misbehaviour probability maps of DeepHyperion (see Figure 5.9b) and DeepHyperion-CS (see Figure 5.9a) for MNIST show well-characterised regions of the feature space that are likely to expose failures, e.g. continuous and thick digits. Therefore, our feature maps can be a powerful tool for developers to understand the conditions responsible for misbehaviours, similarly to the more traditional root-cause analysis.

The probability maps produced by the two tools for the `Mov-Lum` feature combination are similar, with DeepHyperion-CS's map containing slightly more dark cells and 2 more cells with thick border; this is expected, since the two tools achieved similar Mapped Misbehaviours and Misbehaviour Sparseness (see Figure 5.8).

For what concerns `Or-Lum` and `Or-Mov`, DeepHyperion-CS's misbehaviour probability maps are more informative (i.e. have less empty cells) than DeepHyperion's ones. Consistently with the boxplots in Figure 5.8, DeepHyperion-CS's maps contain more dark cells than DeepHyperion's maps.

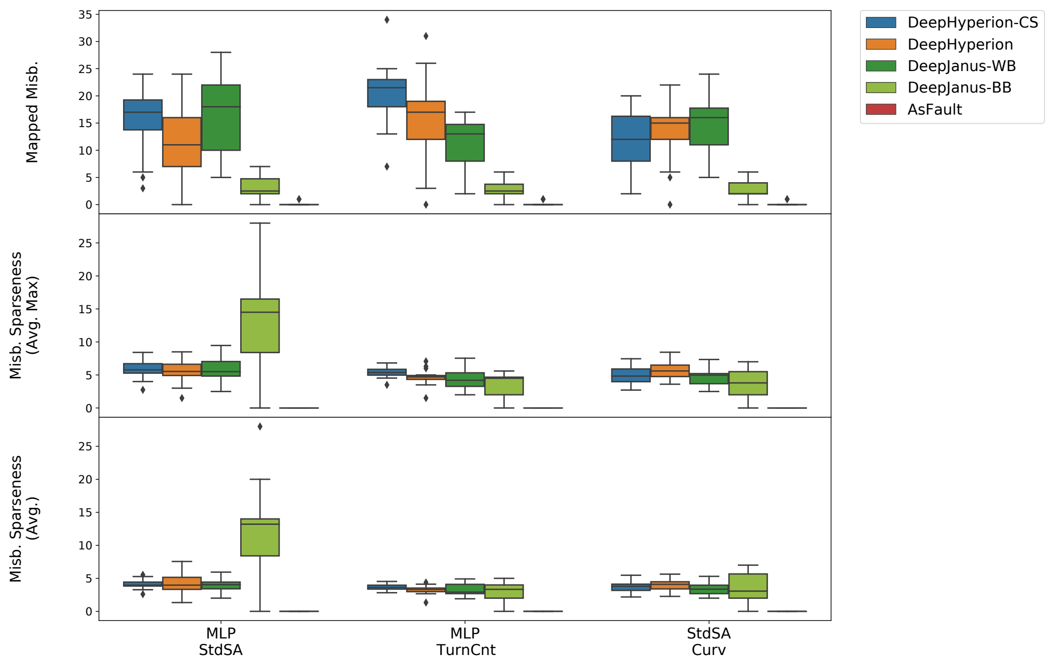


Figure 5.10. RQ1: Mapped misbehaviours found on BeamNG by the considered tools (top) and their sparseness (middle and bottom).

Additionally, thanks to this visualisation we can easily identify the regions in the feature space that DEEPHYPERION-CS explored and DEEPHYPERION missed. For instance, DEEPHYPERION-CS was able to explore large feature space regions characterised by `Or` values smaller than −60.0 (i.e., the left side of `Or-Lum` and `Or-Mov` maps) that DEEPHYPERION missed. The exploration of those regions paid off, as DEEPHYPERION-CS discovered a massive number of new misbehaviours there.

Figure 5.10 shows the Mapped Misbehaviours and Misbehaviour Sparseness obtained when running the tools against our second subject system, BEAMNG.

DEEPHYPERION-CS exposed several diverse misbehaviours for all three feature combinations (more than 10, on average). In particular, DEEPHYPERION-CS found significantly more mapped misbehaviours than the other tools for `MLP-TurnCnt` (more than 20, on average). The contribution score guidance of DEEPHYPERION-CS led to significant improvements over DEEPHYPERION for the `MLP-StdSA` and `MLP-TurnCnt` feature combinations (p-values < 0.05 with medium and large effect size, respectively).

DEEPHYPERION-CS performed almost always significantly better and never performed worse than the other competitors. In fact, only DEEPHYPERION for `StdSA-Curv` and DEEPJANUS-WB for `MLP-StdSA` and `StdSA-Curv` managed to achieve results comparable to DEEPHYPERION-CS (p-values > 0.05), whereas DEEPHYPERION-CS found a significantly higher number of mapped misbehaviours in all the other comparisons (p-values < 0.05, large effect size).

DeepJanus-WB reported significantly more mapped misbehaviours than DeepJanus-BB and proved to be a valid challenger to DeepHyperion-CS in two feature combinations out of three. Instead, AsFault reported almost no misbehaviour across all its runs, which suggests that it might be better suited for testing lane keeping systems at higher speeds and on longer roads than the ones considered in our experimental configuration (we divided the long roads generated by AsFault into segments to make them comparable to those generated by the other tools).

Misbehaviour sparseness metrics (see middle and bottom of Figure 5.10) show a similar trend. For `MLP-TurnCnt`, DeepHyperion-CS generated significantly sparser misbehaviours than all the other tools considering Average Max Misbehaviour Sparseness, whereas it has a comparable misbehaviour sparseness to DeepJanus-BB in terms of Average Misbehaviour Sparseness.

For `StdSA-Curv` and `MLP-StdSA`, DeepHyperion-CS's sparseness is higher than AsFault, comparable to DeepHyperion and DeepJanus-WB, but significantly lower than DeepJanus-BB for both misbehaviour sparseness metrics.

This result can be explained by considering the relatively small number of mapped misbehaviours reported by DeepJanus-BB and their distribution on distant places of the subject's behavioural frontier, which inflate the resulting Misbehaviour Sparseness metric.

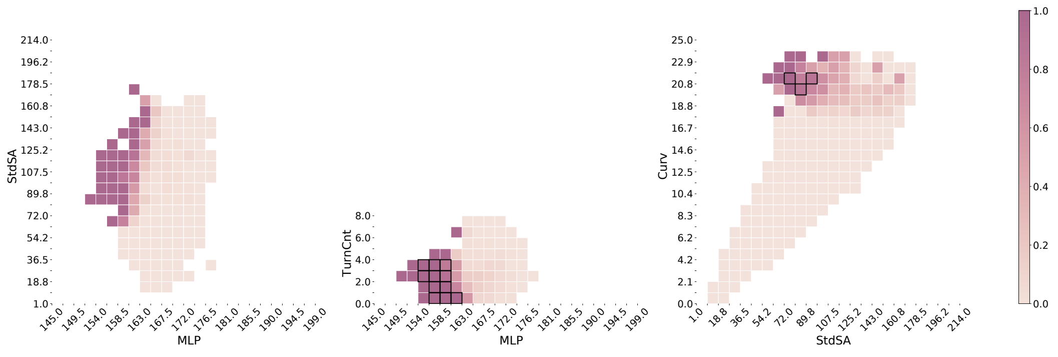
The misbehaviour probability maps of DeepHyperion (Figure 5.11a) and DeepHyperion-CS (Figure 5.11b) show almost the same pattern. The probability maps show also that both tools were able to trigger different test outcomes even when the same behavioural feature is exhibited by the driving agent: in the leftmost maps, the same values of the standard deviation of steering angle (StdSA) feature may or may not trigger misbehaviours, depending on the value of the mean lateral position feature.

Summary: DeepHyperion-CS *can find diverse misbehaviour-inducing inputs for all feature combinations, detecting up to* 100 *more misbehaviours than its best competitor on* MNIST. *The guidance offered by contribution score significantly improved* DeepHyperion*'s effectiveness for* 5 *out of* 6 *feature combinations across both test subjects.*

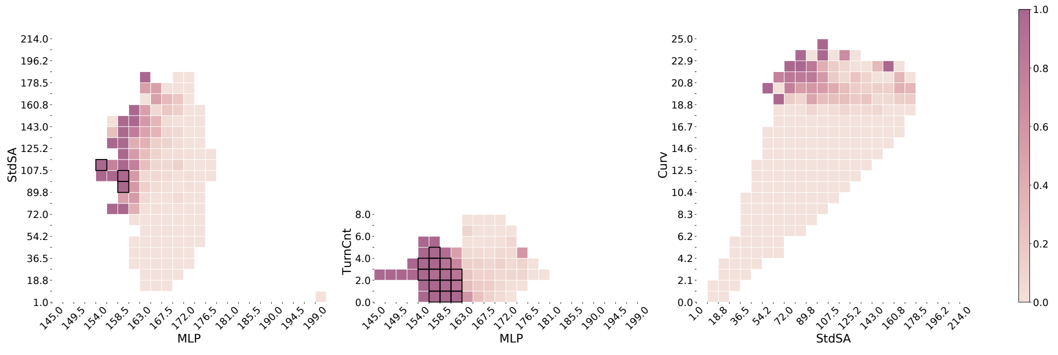
5.3.4 RQ2: Search Exploration

RQ2 investigates the generated tests' adequacy in terms of their feature map coverage (i.e., *Filled Cells*) and diversity (i.e., *Coverage Sparseness*). Figure 5.12 reports the results for MNIST as box plots grouped by feature combination.

For DeepJanus and DLFuzz, we report the cells filled by the inputs returned at the end of the run (i.e., black box) and the number of all the cells filled during the same run (i.e., white box) as two separate boxes. DeepJanus finds pairs of similar inputs that trigger different behaviours (expected vs misbehaviours); since inputs within



(a) DeepHyperion



(b) DeepHyperion-CS

Figure 5.11. Misbehaviour probability maps generated by DeepHyperion (a) and DeepHyperion-CS (b) for BeamNG

pairs are likely to occupy the same cell, DEEPJANUS's Filled Cells values are close to the corresponding Mapped Misbehaviours reported in Figure 5.8. DLFUZZ-BB shows the same values reported for DLFUZZ in Figure 5.8 since this tool returns only misbehaviours. Instead, DLFUZZ-WB has higher values because it includes also the correctly behaving inputs produced during each run.

Figure 5.12 (top) shows that DEEPHYPERION-CS covered all feature maps significantly more extensively than the other tools (with p-values < 0.05 and effect size which is small for `Mov-Lum` and large for `Or-Lum` and `Or-Mov`).

Similarly to RQ1, DEEPHYPERION-CS produced the best results for the `Or-Mov` feature combination, almost doubling the coverage achieved by DEEPHYPERION and tripling the one achieved by DEEPJANUS-WB.

The sparseness metrics reported in Figure 5.12 show that DEEPHYPERION-CS produced significantly sparser inputs than all the other tools for two feature combinations out of three (i.e., `Or-Lum` and `Or-Mov`) with p-values < 0.05 and large effect size. For `Mov-Lum`, DEEPHYPERION-CS achieved a level of coverage sparseness comparable to DEEPHYPERION (p-values > 0.05), but significantly higher than all the other tools.

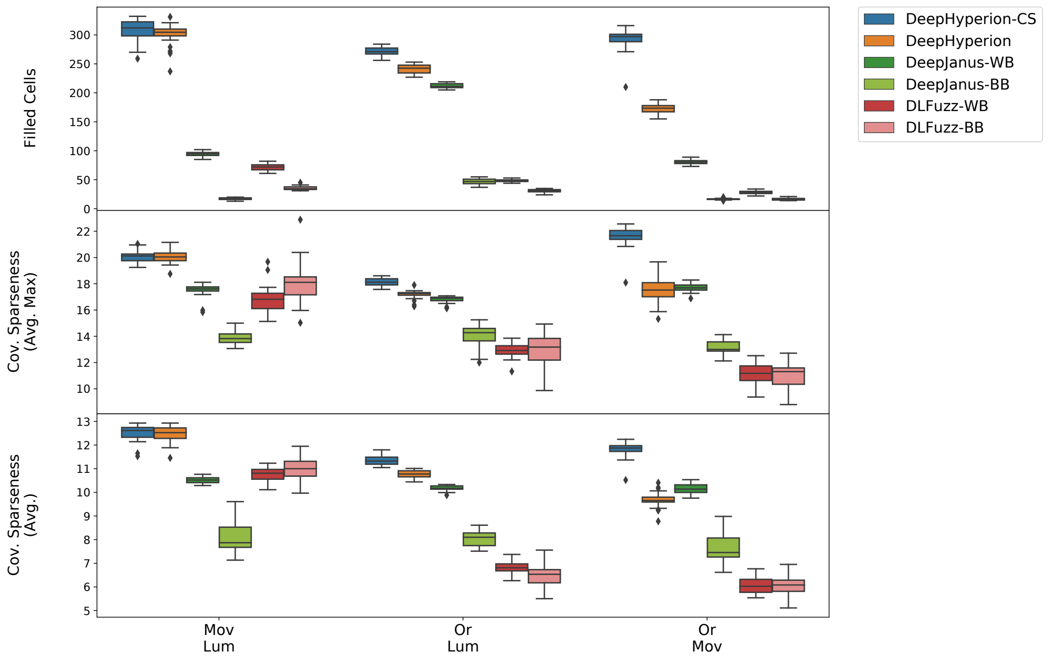


Figure 5.12. RQ2: Filled cells (top) and coverage sparseness (middle and bottom) achieved by the considered tools on MNIST.

Figure 5.13 reports the Filled Cells and the Coverage Sparseness achieved by the tools on the BeamNG subject system. Figure 5.13 (top) shows that DeepHyperion-CS was again particularly good in covering the feature maps. In particular, for the `StdSA-Curv` feature combination, it achieved significantly higher coverage than DeepHyperion (p-value < 0.05, medium effect size). For the other feature combinations, it behaved comparably to DeepHyperion (p-values > 0.05) but filled significantly more cells than all the other tools (p-values < 0.05, large effect size).

As shown in Figure 5.13 both sparseness metrics show that DeepHyperion-CS produced tests that are significantly sparser than DeepJanus-WB, comparably sparse as DeepHyperion, but less sparse than DeepJanus-BB and AsFault.

This result is due to the low feature map coverage achieved by DeepJanus-BB and AsFault, which amplifies the relative sparseness of the (few) filled cells.

Summary: *Our illumination based test generators (i.e.,* DeepHyperion-CS *and* DeepHyperion*) always explored the feature space more extensively than the other tools (up to* 3× *more for* MNIST*). The guidance provided by Contribution Score allowed* DeepHyperion-CS *to fill significantly more cells than* DeepHyperion *for the vast majority of feature combinations (i.e.* 4 *out of* 6*).*

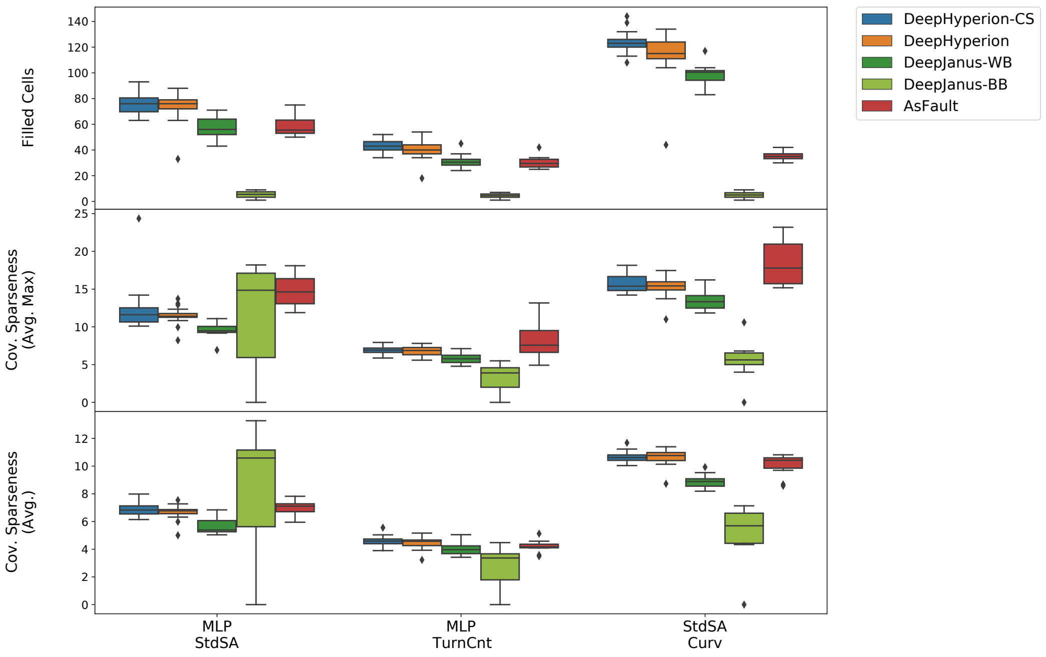


Figure 5.13. RQ2: Filled cells (top) and coverage sparseness (middle and bottom) achieved by the considered tools on BeamNG.

RQ3: Efficiency

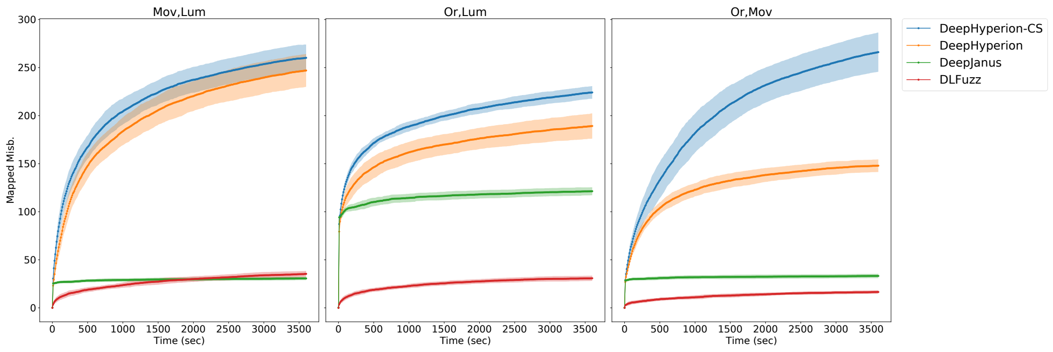
RQ3 investigates DEEPHYPERION-CS's efficiency by analysing the cumulative mapped misbehaviours (Figures 5.14a and 5.15a) and filled cells (Figures 5.14b and 5.15b) for MNIST and BEAMNG. We visualise the evolution of mapped misbehaviours and filled cells throughout the runs by plotting their average values over time as solid lines, surrounded by their standard deviation as transparent shade.

We consider the AUC for mapped misbehaviours and filled cells to quantitatively compare the considered tools' efficiency, i.e., the larger the AUC, the faster the metric increases to high values. For this RQ, we did not consider white box and black box performance separately since we can measure the evolution over time only when white box information is collected during the run.

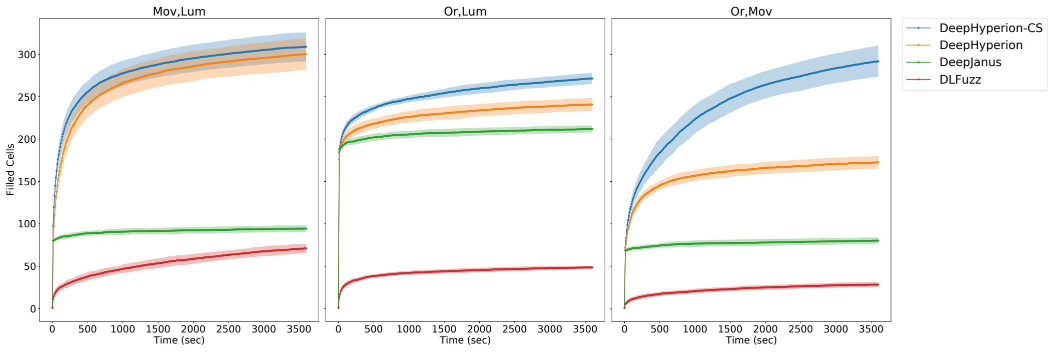
For MNIST, Figure 5.14 shows that DEEPHYPERION-CS achieved a significantly greater AUC than all the other tools for both metrics (p-values < 0.05 and large effect size, with the only exception of mapped misbehaviours AUC for `Mov-Lum` vs DEEPHYPERION, in which the effect size is medium).

In particular, for `Or-Mov`, DEEPHYPERION-CS produced a higher number of mapped misbehaviours in remarkably less time than the other tools (AUC for Mapped Misbehaviours is 65% larger than the second best).

Figure 5.14 also shows that it took a small part of the 1-hour budget (i.e., less than



(a) Mapped Misbehaviours over time for DeepHyperion-CS, DeepHyperion, DeepJanus and DLFuzz on MNIST



(b) Filled Cells over time for DeepHyperion-CS, DeepHyperion, DeepJanus and DLFuzz on MNIST

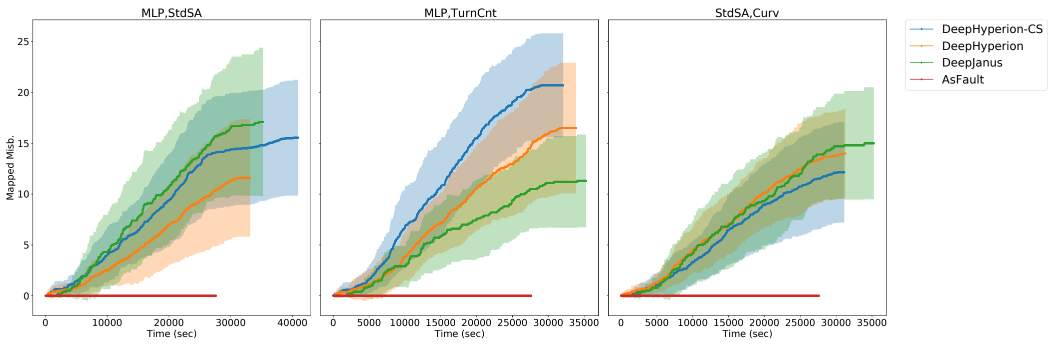
Figure 5.14. RQ3: Filled Cells and Mapped Misbehaviours over time for DeepHyperion-CS, DeepHyperion, DeepJanus and DLFuzz on MNIST (shadows indicate standard deviations for each tool)

three minutes) for DeepHyperion-CS and DeepHyperion to outperform the other tools, by generating a significantly higher number of misbehaviours and filled cells for all feature combinations (p-values < 0.05 and large effect size).

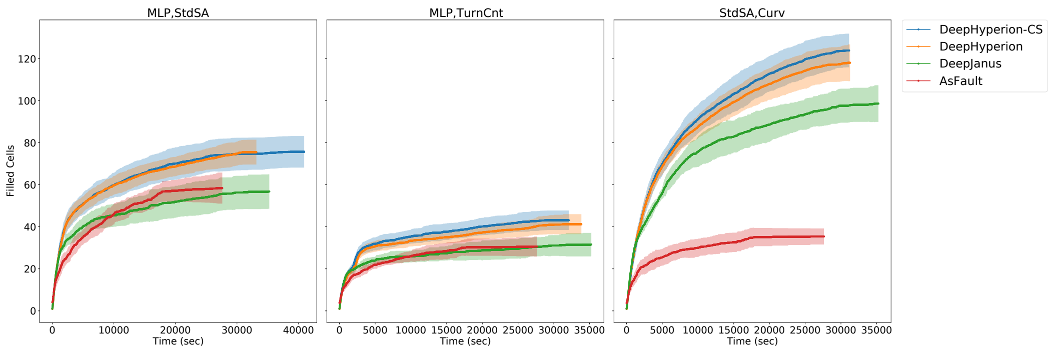
By comparing the results achieved over time by DeepHyperion-CS and DeepHyperion, we can notice that DeepHyperion-CS dramatically outperformed DeepHyperion after only three minutes for `Or-Lum` and eight minutes for `Or-Mov`, whereas they followed a similar trend for `Mov-Lum`.

These results confirm that DeepHyperion-CS is extremely efficient from the very beginning of the search process in exploring the feature space and exposing diverse misbehaviours in MNIST. Moreover, DeepHyperion-CS kept discovering new cells, although at a lower pace as time progresses, which suggests that it did not reach saturation within the given time budget.

Figure 5.15 shows the evolution of mapped misbehaviours and filled cells for



(a) Mapped Misbehaviours over time for DeepHyperion-CS, DeepHyperion and DeepJanus on BeamNG



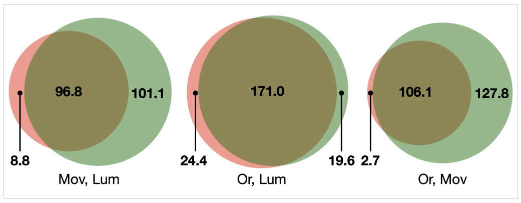
(b) Filled Cells over time for DeepHyperion-CS, DeepHyperion, DeepJanus and AsFault on BeamNG

Figure 5.15. RQ3: Filled Cells and Mapped Misbehaviours over time for DeepHyperion-CS, DeepHyperion, DeepJanus and AsFault on BeamNG (shadows indicate standard deviations for each tool)

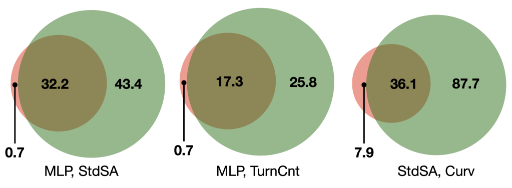
BeamNG. While we assigned all tools a budget of 10 hours of *simulation* time, in this RQ we are interested in assessing their practical efficiency and, thus, we compute the evolution of the considered metrics over *real* time. As a consequence, the results span across different time ranges for each tool. To guarantee a fair comparison, we compute the AUC by considering the minimum run time of all the tools.

Figure 5.15a shows that DeepHyperion-CS was significantly more efficient in finding diverse misbehaviours than all the other tools for `MLP-TurnCnt` (higher AUC, with p-values < 0.05, large effect size), while it never performed worse than the competitors for the other feature combinations: for `StdSA-Curv`, DeepHyperion-CS achieved AUC of mapped misbehaviours comparable to DeepHyperion and DeepJanus, and significantly better than AsFault; whereas only DeepJanus was as efficient as DeepHyperion-CS for `MLP-StdSA`.

As regards AUC of filled cells, Figure 5.15b shows that DeepHyperion and DeepHyperion-CS covered the feature maps significantly more efficiently than DeepJanus and AsFault



(a) MNIST



(b) BeamNG

Figure 5.16. RQ4: Average Filled Cells Expansion achieved by DeepHyperion-CS (green/right circles) over the training set (red/left circles).

for all feature combinations (p-values < 0.05 and large effect size).

In particular, DeepHyperion-CS showed significantly better efficiency in filling cells than DeepHyperion for `MLP-TurnCnt` (p-value < 0.05, medium effect size), whereas, they achieved comparable efficiency for the other two feature combinations (p-values > 0.05).

Summary: DeepHyperion-CS *was extremely efficient, as it increasingly explored the feature space throughout the time budget and it found misbehaviours within the first few minutes of exploration.* DeepHyperion-CS *was always significantly more efficient than the competitors on* MNIST. *On* BeamNG, DeepHyperion-CS *showed either significantly higher or comparable efficiency in comparison with the other tools. The guidance of the contribution score remarkably improved efficiency (in* 9 *out of* 12 *comparisons against* DeepHyperion*).*

RQ4: Training Data Expansion

RQ4 studies the relationship between the data used for training the DL system under test and the test inputs generated by DeepHyperion-CS, by identifying features that were under-represented in the training set or were not associated with any misbehaviour.

Venn diagrams in Figure 5.16a and Figure 5.16b illustrate the Filled Cell Expansion achieved by DeepHyperion-CS over the training set for each feature combination. The red circles (left) represent the cells filled by the training set, the green circles (right) represent the ones filled by DeepHyperion-CS's generated inputs, and the overlapping region represent cells that are covered by both.

DeepHyperion-CS achieved a remarkable filled cell expansion for the BeamNG system (see Figure 5.16b), where not only it filled most cells already covered by training data, but it also explored new uncovered regions in the feature maps, especially for the `MLP-StdSA` and `MLP-TurnCnt` feature combinations.

Figure 5.16a shows that DeepHyperion-CS was also able to improve the MNIST initial training set by adding samples that better cover some feature combinations (see,

Table 5.3. RQ4: Mapped misbehaviour expansion achieved by DeepHyperion-CS over the training set

Subject	**Feature Combination**	**Mapped Misbehaviour Expansion**	
		MME_{uncov}	MME_{cov}
MNIST	(Mov, Lum)	101.1 ± 12.0	60.9 ± 3.4
	(Or, Lum)	19.7 ± 4.9	19.6 ± 2.4
	(Or, Mov)	127.8 ± 16.5	48.3 ± 2.2
BeamNG	(MLP, StdSA)	12.4 ± 5.1	3.1 ± 2.1
	(MLP, TurnCnt)	11.2 ± 3.9	9.5 ± 2.6
	(StdSA, Curv)	12.1 ± 4.9	0.0 ± 0.0

e.g., the Venn diagram for `Mov-Lum`). This task was not trivial since the MNIST training set by LeCun et al. [82] has been carefully crafted to be representative of its domain.

Table 5.3 summarises the average mapped misbehaviours expansion achieved by DeepHyperion-CS. Specifically, column MME_{uncov} reports the mapped misbehaviours that DeepHyperion-CS found in cells that were not covered by the training set, while column MME_{cov} reports the mapped misbehaviours found by DeepHyperion-CS in cells already covered by correctly behaving training inputs.

The results show that DeepHyperion-CS was always able to find new misbehaviour-inducing feature combination values in cells that were either uncovered or already covered by the training set. In particular, DeepHyperion-CS found misbehaviours in cells that did not expose any issue in the training set for 5 out of 6 feature combinations.

Each MME_{uncov} value is generally higher than the corresponding MME_{cov} for the same feature combination. This indicates that cells covered by correctly behaving training inputs could still trigger misbehaviours, but such misbehaviours are harder to expose.

Summary: DeepHyperion-CS *was able to expand the initial training data for all feature combinations, achieving up to 200% more filled cells for the* `StdSA-Curv` *feature combination. Moreover,* DeepHyperion-CS *not only found new misbehaviour-inducing feature combination values in cells that were uncovered by the training set, but often also in covered ones.*

Threats to Validity

Construct Validity: The performance of the proposed approach and the quality of its results depend on the selected features and the procedures to quantify them. For instance, there is the risk that the adopted metrics do not accurately quantify the selected features. Moreover, the relevance of the features depends on the assessors' knowledge of the domain and the representativeness of the data used to extract the features. To mitigate this threat, we followed a systematic procedure to identify relevant features and utilised a well established statistical correlation analysis to check that the adopted metrics can quantify them. In particular, this procedure involved assessors that are experts on testing DL systems and relied on large dataset of 630 images for MNIST and 440 virtual roads for BeamNG.

Internal Validity: To limit as much as possible the chance that the comparison between DeepHyperion and DeepHyperion-CS was affected by other con-causes, we used the same code base for both tools. As a result, we provide a unique framework in which the users can control the level of contribution-based selection pressure on the overall search process, i.e., the users can easily use the original DeepHyperion by setting the hyper-parameter controlling the probability of using the contribution score selection to 0. A threat that could affect the experimental comparison against existing test input generators is that their purpose is different from illuminating the feature space. Therefore, their output may contain only the most critical inputs but exclude interesting inputs found during their runs. We addressed this threat by considering also the inputs generated (and possibly discarded) during the input generation process, in addition to the ones reported as final result by the tools. Furthermore, the internal validity of the open coding may be threatened by elements that may introduce inconsistencies in the assessors' evaluations independently of the data, such as the data order and the repetitiveness of the task. To mitigate this threat, we conducted a pilot study in which multiple assessors evaluated the same images, presented in a randomised order. Moreover, we granted the assessors a generous time budget to perform this task (i.e., one month) through our Web application, which allowed them to interrupt the task whenever they felt tired and resume it later on.

External Validity: The choice of subject DL systems is a possible threat to the external validity. To mitigate this threat, we chose two DL systems which solve two different problems, i.e., MNIST solves a classification problem, while BeamNG is a self-driving car software that solves a regression problem. We considered DL architectures which are widely used in the literature and regarded as state of the art. Moreover, we adopted standard training procedures and validated them with standard performance metrics, i.e. classification accuracy and mean squared error. In comparison to the original DeepHyperion's experimental setup [159], in this work we extend the generalisability of the results by considering another state-of-the-art test generator, i.e., AsFault. The choice of relevant features introduces another threat to external validity as DeepHyperion might not identify misbehaviours that do not align with the selected features. Further studies involving more test subjects and domain experts, including practitioners from industry, should be carried out to fully assess the generalisability of our findings and the impact of the feature selection.

Conclusion Validity: Random variations might have affected the results, given the highly stochastic nature of both DL systems and the considered test generators. We mitigated this threat by following the widely adopted guidelines for comparing randomised test generation algorithms proposed by Arcuri and Briand [5]. In particular, we used a generous budget (i.e., multiple, long runs) and assessed the results' significance through standard statistical tests.

5.4 Conclusion

DeepHyperion-CS has demonstrated to be able to explore the DL systems' feature space at large and trigger diverse misbehaviours thanks to its illumination search based algorithm.

Our empirical study showed that DeepHyperion-CS is more effective than state-of-the-art DL testing tools in generating failure-inducing inputs associated with highly diverse features. In the reverse direction, we showed that DeepHyperion-CS is useful to detect the feature combinations that are most likely to induce a system misbehaviour. Moreover, we provided evidence that the inputs generated by DeepHyperion-CS can be also useful for characterising and expanding the datasets used to train the DL system.

5.5 Reproducibility

The code implementing DeepHyperion-CS, the dataset, and all the scripts to replicate the experimental evaluation are available online [160].

Chapter 6

Focused Test Generation for Deep Learning Systems

A feature map depicts the feature space defined by N relevant dimensions of variation (i.e., the map axes, each corresponding to an input feature). Test inputs are placed in a feature map based on their feature values. Figure 6.1 shows an instance of bi-dimensional feature map for a classifier of handwritten digits from the MNIST database [81]. This feature map is defined by the digit rotation (Orientation) and the stroke's boldness (Luminosity) and represents, for each combination of feature values (a map cell), the corresponding probability of exposing a misbehaviour, i.e., darker colors correspond to higher probabilities.

At testing time, feature maps highlight areas of the feature space that are not adequately covered [13], while, during operation, critical feature values may be observed that are under-represented in the train/test datasets used at development time. For them, new and diverse input data need to be collected and labelled manually [49]. Testers will try to find multiple misbehaviour-inducing inputs, focusing on specific feature combinations, as these additional inputs can be used to improve the DL system quality delivered to production, by fine tuning the DL models on such new data.

We introduced a novel way to generate misbehaviour-inducing inputs with specific, user-defined feature values. Our approach is the first focused input generator for DL based systems targeting human-interpretable features. It can be employed to collect new diverse inputs with critical, misbehaviour-inducing characteristics (❶ in Figure 6.1), to stress the system to expose failures with inputs that do not seem critical (❷), or to generate new data with underrepresented or unseen feature values (❸). An example of usage scenario is a DL system that, once deployed in operation, has to handle frequently feature combinations never (❸) or rarely (❶, ❷) observed at development time (this is also called the development to operation, *dev2op*, shift [49]).

To test such feature combinations, we propose DEEPATASH, a search-based focused test generator for DL systems. DEEPATASH can be configured with alternative search

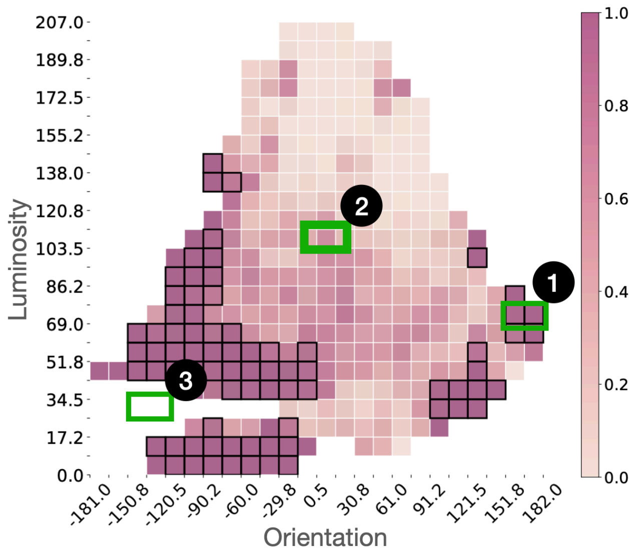


Figure 6.1. Feature map for a handwritten digit classifier. The axes quantify orientation and luminosity of the digits. The cells report the probability of exposing a misbehaviour for the corresponding feature value combinations, i.e., darker colors correspond to higher misbehaviour probabilities.

strategies (single or multi-objective) and sparseness metrics. It takes as input the desired target feature value ranges and it optimizes both the generated input sparseness and the input closeness to the target in the feature map.

We evaluated DEEPATASH on two different classification problems (recognition of handwritten digits and sentiment analysis of movie reviews). For both problems, results show that DEEPATASH is effective at generating diverse failure-inducing test inputs within the target feature map cell in different usage scenarios. The inputs generated by DEEPATASH have been used to fine-tune the DL models under study and improve their performance on under-represented feature combinations, which were initially not handled at all (0% accuracy) and reached approximately 99% accuracy after fine-tuning, with no regressions.

However, DEEPATASH may face challenges in generating diverse inputs for systems with higher evaluation costs, such as ADSs. To address this issue in resource-intensive case studies, we introduce DEEPATASH-LR. This extension integrates DEEPATASH with a surrogate model, predicting misbehavior likelihood and behavioral features, optimizing computational resources and enhancing the use of the test generation budget. Evaluation in the context of a LKAS DL component shows that DEEPATASH-LR, with its surrogate model, effectively generates misbehavior-inducing inputs, allowing fine-tuning of the ADS and significant performance improvements without notable regressions.

6.1 The DeepAtash Technique

DeepAtash aims to generate misbehaviour-inducing test inputs with characteristics defined by the user, i.e., inputs belonging to a predefined feature map cell that trigger an unexpected behaviour. As a secondary goal, it maximises the sparseness among the generated solutions to obtain diverse inputs. Given the desired target ranges of feature values, referred to as the *target cell* (e.g., $[1:5] \times [10:15]$ if we want the first feature f_1 to be between 1 and 5 and the second f_2 between 10 and 15), DeepAtash directs the generation of new inputs toward the feature subspace defined by these values. DeepAtash adopts evolutionary search to generate inputs that: (1) are close to the target cell; (2) are diverse from the already found solutions; and (3) trigger a misbehaviour of the DL system. It iteratively manipulates an initial set of inputs (called *seeds*) until they fall into or near to the target cell. The evolution is guided by fitness functions representing the closeness to misbehaviour, the distance to the target cell and the distance from the previously found solutions.

Algorithm 2 outlines the high-level steps of our focused test input generation technique. The algorithm starts by initialising an empty archive A (line 1), which will store the best test inputs generated during the search, i.e., the most sparse inputs with feature values inside or close to the target ranges.

Function InitialisePopulation (line 2) instantiates an initial population P with the desired number of individuals (*popsize*), by drawing elements from an initial pool of seeds S provided as input. Usually, S is a subset of the test set available with the DL system under test. The warm-up phase is completeted by determining the fitness values of all the individuals of the initial population (line 3).

The main evolutionary loop is performed until the termination condition is satisfied (lines 4-13). At each iteration, the population is mutated by genetic operators to produce its offspring Q (lines 5-8). The worst individuals of the population are replaced by the Repopulation operator, which generates new inputs from the initial pool of seeds S (line 9). Repopulation takes as input the archive A to avoid selecting seeds already used to produce individuals stored in the current archive A.

Function Evaluate calculates the fitness of the current population P and its offspring Q (line 10). The inputs close to the target cell, i.e., those whose distance from the target cell is smaller than a threshold, are stored in the archive A, if they are better than the previously discovered solutions (line 11). Then, the *popsize* fittest individuals are selected for the next generation by the Select function (line 12). When optimizing multiple fitness functions at the same time, ranking of individuals for selection is based on Pareto dominance and crowding distance, as prescribed by the NSGA-II multi-objective optimization algorithm [23]. When the execution budget B is elapsed, the algorithm returns the misbehaviour-inducing inputs stored in the archive as final outcome (lines 14-15).

In the rest of this Section, we describe the key aspects of DeepAtash and how we

Algorithm 2: DEEPATASH's Focused Test Generation

```
Input  : $B$: execution budget
         $targetCell$: target feature value ranges
         $archivesize$: target archive size
         $S$: set of input seeds
         $popsize$: population size
Output : $A$: archive of test inputs in the target cell
1  $A \leftarrow \emptyset$;
2  $population\ P \leftarrow$ INITIALISEPOPULATION($S$, $popsize$);
3  EVALUATE($P$, $A$, $targetCell$);
4  while $elapsedBudget < B$ do
5      $offspring\ Q \leftarrow P$ ;
6      foreach $q \in Q$ do
7          $q \leftarrow$ MUTATE($q$) ;
8      end
       // substitute the worst individuals
9      $P \leftarrow$ REPOPULATION($P$, $S$, $A$);
10     EVALUATE($P \cup Q$, $A$, $targetCell$);
11     $A \leftarrow$ UPDATEARCHIVE($P \cup Q$, $archiveSize$, $targetCell$);
12     $P \leftarrow$ SELECT($P \cup Q$, $popsize$);
13 end
14 $A \leftarrow$ FILTERMISBEHAVIOURS($A$);
15 return ($A$)
```

applied it to the handwritten digit recognition and movie review sentiment analysis tasks.

6.1.1 Input Representation

DEEPATASH performs semantic-based input generation, i.e., it leverages semantic information about the inputs (e.g., digit shape or sentiment polarity of a word), rather than simply corrupting them (e.g., changing pixel values or modifying letters in a word). Examples of semantic-based approaches are model-based techniques, which are standard practice in several domains, including safety-critical ones such as automotive [146, 79]. Semantic-based test input generation has been already successfully applied to DL system testing [1, 2, 3, 119, 159, 117]. In general, it is applicable to any domain for which the semantic of the input data can be modeled. For this reason, in this work we consider two domains for which semantic models are available: handwritten digit recognition and movie review sentiment analysis.

For *handwritten digit recognition*, test inputs are images in the MNIST database format

[81]. In particular, digits are encoded as 28 × 28 images with greyscale levels that range from 0 to 255. DEEPATASH models each digit as a sequence of control points that define a Bézier curve, according to the Scalable Vector Graphics (SVG) representation. To this aim, DEEPATASH leverages the operations performed by the Potrace algorithm [129], which vectorises a binary image by drawing a smooth contour made of Bézier segments.

For *movie review sentiment analysis*, test inputs are texts from the IMDB database [92]. DEEPATASH represents each text as a tokenised padded sequence with a predefined length, i.e., a tokeniser converts text inputs to the corresponding sequence of tokens and then applies padding to have vectors of the same length. DEEPATASH obtains the semantic information of a text by associating each of its words to the corresponding polarity, obtained from the English Opinion Lexicon [59] which contains a list of words with positive and negative polarity. The words that are neither positive nor negative are considered neutral.

6.1.2 Fitness Functions

We use three fitness functions to guide DEEPATASH's focused generation. They quantify: (1) the distance of the test input from the target cell; (2) the closeness of the DL system to exhibiting a misbehaviour when executing the given test input; and, (3) the distance of the input from the previously found solutions (i.e., its sparseness).

Distance from the Target Cell

To measure the distance of an individual x from the target cell c, DEEPATASH computes the Manhattan distance between the cell containing the individual x and the target cell. This fitness function is minimised.

$$\min \mathit{fitness}_1(x) = \min \mathit{dist}(x,c) \tag{6.1}$$

Given a target cell $c = [l_1 : u_1] \times \ldots \times [l_N : u_N]$, with N the number of features being considered (usually 2), the range size $s_i = u_i - l_i$ along each dimension f_i (with $i \in \{1,\ldots,N\}$) determines the Manhattan distance of a given individual x from the target cell c, according to the following equations:

$$d(x_i,c_i) = \begin{cases} \left\lceil \frac{l_i - x.f_i}{s_i} \right\rceil, \text{ if } x.f_i < l_i \\ 0, \text{ if } l_i \leq x.f_i < u_i \\ \left\lceil \frac{x.f_i - u_i}{s_i} \right\rceil, \text{ if } x.f_i > u_i \\ 1, \text{ if } x.f_i = u_i \end{cases} \tag{6.2}$$

$$d(x,c) = \sum_{i=1}^{N} d(x_i,c_i) \tag{6.3}$$

Along each dimension i, the difference between the individual's coordinate $x.f_i$ and the cell's lower/upper bound (l_i or u_i), divided by the cell size s_i, gives the number of cells that separate x and c along f_i (the value is rounded up, to get an integer). The sum of the number of separating cells across all dimensions corresponds to the Manhattan distance between x and c. Let us consider for example a target cell $c = [2:6] \times [6:8]$ and a candidate solution x whose feature values are $x.f_1 = 8$, $x.f_2 = 3$. The Manhattan distance between x and c is hence $\lceil(8-6)/4\rceil + \lceil(6-3)/2\rceil = 1 + 2 = 3$.

Closeness to Misbehaviour

DeepAtash aims to generate test inputs that trigger misbehaviours of the DL system under test. Therefore, it promotes inputs that are more likely to trigger a misbehaviour by minimising a problem-specific fitness function which measures how close the DL system is to misbehave, when exercised with the evaluated input.

$$\min \mathit{fitness}_2(x) = \min \mathit{evaluateBehaviour}(x) \tag{6.4}$$

For the *handwritten digit recognition* problem, we exploit the activation levels of the classifier's output softmax layer, since they can be interpreted as a confidence level assigned to each of the possible classes [42], i.e., the predicted class corresponds to the one with highest confidence. As a fitness function, DeepAtash considers the difference between the confidence level associated to the expected class (which corresponds to the expected behaviour) and the maximum confidence level associated to any other class. In this way, the fitness value decreases when the system becomes less confident towards the expected class and more confident towards one of the other classes, while it assumes a negative value when the input is misclassified.

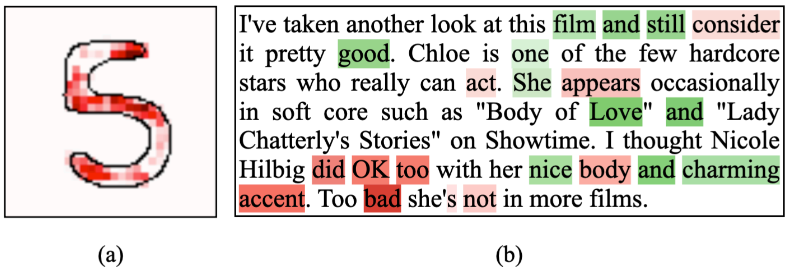
The *movie review sentiment analysis* problem has two classes, i.e., negative and positive sentiments. Therefore, we consider the fitness as the difference between the confidence level associated to the expected class and the one associated to the other class.

Sparseness

An effective focused test input generator should ensure that the inputs found are different among them, thus providing a richer set of execution scenarios than a mere repetition of the same one. To achieve this goal, DeepAtash maximises a fitness function which measures how different an input is from the solutions already found during the search. More specifically, DeepAtash computes the sparseness of the individual x with respect to the ones in the archive A as follows:

$$\max \mathit{fitness}_3(x) = \max \mathit{spars}(x, A) \tag{6.5}$$

Function *spars* measures the minimum distance of x from the solutions in the archive A: $\min_{y \in A} \mathrm{dist}(x, y)$. The distance function *dist* is computed on pairs of inputs and is domain-specific.



I've taken another look at this film and still consider it pretty good. Chloe is one of the few hardcore stars who really can act. She appears occasionally in soft core such as "Body of Love" and "Lady Chatterly's Stories" on Showtime. I thought Nicole Hilbig did OK too with her nice body and charming accent. Too bad she's not in more films.

Figure 6.2. Explanatory heatmaps generated by the Integrated Gradients technique. (a) A heatmap for a sample digit 5 is shown, with red pixels highlighting the parts of the digit that contribute more to the predicted label; (b) A heatmap for sample text is presented, green shades highlight the words contributing to the positive sentiment, while red shades highlight the words contributing to the negative sentiment.

Since different distance functions may lead to different results, for the *digit recognition* and *movie review sentiment analysis* problems, we considered alternative metrics: input space, explanation space, and latent space sparseness.

Input space sparseness measures distances between inputs in the space defined by the input elements. For *handwritten digit recognition*, it is computed as the Euclidean distance between pairs of image vectors. This metric is the most widely used in the literature due to its simplicity and has been already successfully applied to test image-based DL systems [119, 159, 51]. For *movie review sentiment analysis*, sparseness is computed as the Levenshtein distance between pairs of text input strings [86, 113, 149].

Computing distances in such high-dimensional spaces is inefficient and suffers from scalability problems. High-dimensional and sparse spaces naturally hinder the search from finding similarity between data and contain information that is not relevant to the prediction of the DL system, i.e. they are affected by the *curse of dimensionality* [11].

Explanation sparseness leverages *Integrated Gradients* [139], an explainable AI technique [144], which highlights the pixels/words of the original image/text that contribute the most to the DL system's prediction in a so-called *heatmap*, as in the one shown in Figure 6.2. Then, we compute the Euclidean distance between pairs of heatmaps. While handwritten digit images have the same size, movie reviews may have different lengths. Therefore, we generate a vector of size S, corresponding to the size of the vocabulary used by the tokeniser, where each vector component e_i is the contribution value of the i-th word.

Explanation sparseness is still based on the original, high dimensional input space, but it focuses on the relevant part of the inputs by replacing input values with heatmap values.

Latent space sparseness measures distances between inputs in the latent space. For digit recognition, it is defined on the latent vectors produced by an autoencoder. Autoencoders are neural networks trained to learn a representation of the input data (the *encoding* or *latent space*) that has a lower dimensionality than the original input space, but retains most of the original information and discards noise. In this way, it is still possible to reconstruct the input in the original input space starting from its latent vector. To measure latent space diversity, we train a Variational AutoEncoder (VAE), a particular autoencoder architecture that maps the original image to a latent vector of Gaussian random variables by estimating the mean and the variance of each latent vector variable. To compute the distance between two handwritten digit images, our latent space diversity metric uses the Euclidean distance between the means of the latent vector variables.

For movie review sentiment analysis, the latent space sparseness is defined on the latent vectors generated by a Doc2Vec model [80], which is an unsupervised DL algorithm for representing documents as vectors in a lower-dimensional space. More specifically, Doc2Vec represents each document as a single vector which encapsulates the semantics of the whole document.

Latent space sparseness tackles the issues of high-dimensional and noisy input spaces by focusing on lower-dimensional representations of the relevant information carried by the inputs.

6.1.3 Archive of Solutions

The archive of solutions stores the best individuals encountered during the search and, at the end of the last search iteration, it contains the final solutions. The archive is particularly useful to prevent the *cycling* phenomenon, i.e., when the search moves from one cell of the feature space to another and back again, with no memory of the cells it has already explored [101].

The UPDATEARCHIVE function manages the archive and is described in Algorithm 3. When the archive is not full, all the candidate individuals placed on target or in the neighbouring feature map cells, i.e. those with a distance to the target cell lower than 1, are included into the archive (lines 2-4). Otherwise, if the archive is full, the new candidate input competes with the worst individual in the archive based on their values of $fitness_1$, $fitness_2$ and $fitness_3$. The worst individual in the archive is the one with the highest distance to the target and (for equal distances to the target) the lowest sparseness (line 6). If the candidate individual has lower distance to the target than the worst archived individual, UPDATEARCHIVE replaces the former with the latter within the archive (lines 7-9). When the compared inputs have equal distance to the target, the algorithm evaluates their closeness to misbehaviour and keeps in the archive the best one, which is closer to exposing a misbehaviour (lines 11-15). If the compared individuals have the same distance to the target and to a misbehaviour, they compete on the basis of their sparseness: the sparser one is kept, while the other is discarded (lines 17-19).

Since the archive may contain correctly-behaving inputs, the FILTERMISBEHAVIOURS function is performed at the end of the search to keep only misbehaviour-inducing

Algorithm 3: The UPDATEARCHIVE function

```
Input : $P$: population
        $archiveSize$: target size of the archive A
        $targetCell$: target feature value ranges
1  foreach $p \in P$ do
2      if DIST($p$, $targetCell$) $\leq 1$ then
3          if $A$ is not full and $p \notin A$ then
4              A.insert($p$) ;
5          else
6              $ind \leftarrow$ GETWORSTINDIVIDUAL($A$);
7              if DIST($p$, $targetCell$) $<$ DIST($ind$, $targetCell$) then
8                  $A$.insert($p$) ;
9                  $A$.remove($ind$) ;
10             else
11                 if DIST($p$, $targetCell$) $==$ DIST($ind$, $targetCell$) then
12                     if $p.behaviour < ind.behaviour$ then
13                         $A$.insert($p$) ;
14                         $A$.remove($ind$) ;
15                     else
16                         if $p.behaviour == ind.behaviour$ & $p.sparse > ind.sparse$
                             then
17                             $A$.insert($p$) ;
18                             $A$.remove($ind$) ;
19                         end
20                     end
21                 end
22             end
23         end
24     end
25 end
26 return $A$ ;
```

inputs (see Algorithm 2).

6.1.4 Search Strategies

We evaluated two different search strategies for DEEPATASH: Single-Objective search, which optimizes only the distance to the target, and Multi-Objective search, which explicitly rewards also the closeness to misbehaviour and the sparseness.

Single-Objective Search

As single-objective search strategy, we adopt a Genetic Algorithm (GA) since it previously showed to be very effective for test generation [33]. In particular, we adopt a population-based GA that minimises the Manhattan distance to the target cell. At each iteration, the best individuals in the current population and the offspring are selected, based on their single fitness value, to be part of the next population.

Multi-Objective Search

In this strategy, we cast the focused test generation problem as a multi-objective search problem, by optimising all three fitness functions defined in Section 6.1.2 at the same time. In particular, we adopt the NSGA-II algorithm [23] since it is widely used and it is reported to be very effective in test case generation [78, 96, 109, 2, 119, 117]. NSGA-II applies Pareto front analysis and promotes the solutions that are not dominated by any other individual, i.e., those representing the best trade-offs among the fitness functions. More precisely, a solution x dominates another solution y if x is not worse than y in all fitness values, and x is strictly better than y in at least one fitness value. The final ranking of individuals is based on Pareto dominance (i.e., non dominated fronts are selected and removed from the solutions one after the other) and crowding distance (i.e., within the same Pareto front, distant individuals are selected), as recommended by the NSGA-II multi-objective optimisation algorithm [23]. While this search strategy comes with some overhead, as it computes multiple fitness functions, dominance and crowding distances, it may improve the archived solutions by explicitly promoting diverse and misbehaviour-inducing inputs.

6.1.5 Population Management

DEEPATASH starts its search from an initial population of size *popsize*, which is obtained by randomly choosing from a pool of inputs, named *seeds*. Function INITIALISEPOPULATION (line 2 in Algorithm 2) selects $popsize$ different initial individuals among the seeds.

More specifically, for *handwritten digit recognition*, seeds are all the inputs from the MNIST test set. Instead, for *movie review sentiment analysis*, seeds are the inputs in the IMDB test set that are closer than a predefined threshold *maxDist* to the target cell. This choice is due to the large size of the IMDB test set: we consider only the most promising inputs. We determined *maxDist* after some preliminary DEEPATASH runs with increasing values of *maxDist* (starting from 0) and selected the minimum value that resulted in a reasonable number of archived solutions (see Table 6.1). One common issue in search-based approaches is that the exploration could get stuck in local optima, despite the use of mechanisms to promote diversity such as our fitness function in Equation 6.5. To mitigate this situation and further vary the population, DEEPATASH uses the REPOPULATION operator, which replaces at each iteration a fraction of the worst performing individuals in the current population, i.e., the individuals with the lowest fitness. The new individuals are generated starting from a randomly chosen seed, which is not already represented in the current population and the archive.

This genetic operator can be tuned by setting the *repopulation upperbound* hyperparameter, that determines the range from which the number of individuals to replace is uniformly sampled. As an example, if the repopulation upper bound is set to 100, at each iteration, a number *nrep* is uniformly sampled between 1 and 100, and the *nrep* worst individuals in the current population are replaced by newly generated individuals (see Table 6.1).

6.1.6 Mutation

A new input is obtained from an existing one by applying the MUTATE operator (line 7 in Algorithm 2). This operator applies a perturbation to the original input by leveraging its semantic (i.e., the digit model control points or the word's synonyms). For the *handwritten digit recognition* problem, the mutation operator randomly chooses a control point of the SVG model and applies a displacement to it in the two-dimensional space. For the *movie review sentiment analysis* problem, we defined three mutation operators: (1) replacing a word with its synonym obtained from Wordnet[1]; (2) adding an "and" conjunction after an adjective, followed by a synonym of the adjective; and (3) duplicating a sentence.

Each time an input is mutated, DEEPATASH verifies that the mutant complies with the ad hoc constraints to ensure that the input still belongs to the input validity domain and preserves its original label [120]. When the mutated individual is considered invalid it is discarded and its parent is mutated again. For the *handwritten digit recognition problem*, DEEPATASH verifies that the Euclidean distance between the mutant and the starting seed is greater than 0 and lower than or equal 2. For the *movie review sentiment analysis* problem, the mutated individual is considered invalid if the number of sentiment words differs more than a threshold *sentimentDist* from the initial one. To validate such heuristic constraints, we manually inspected a set of test inputs produced by DEEPATASH and found that in all cases label preservation and validity were confirmed. Therefore, we are confident that misbehavior-inducing inputs are actually producing misbehaviours

[1]https://wordnet.princeton.edu

with high probability. When moving to a different domain, proper heuristic validation functions must be designed for domain-specific mutation operators.

6.2 Experimental Evaluation on Image and Text Classifiers

6.2.1 Research Questions

The goal of our evaluation is to understand the effectiveness of our approach in generating misbehaviour-inducing test inputs with the desired features. In particular, we consider different possible configurations of DEEPATASH, compare it with an existing state-of-the-art test generator (DEEPHYPERION-CS), and investigate the usefulness of the generated test inputs. Therefore, we answer the following research questions:
RQ1 (Effectiveness): *Which* DEEPATASH *configuration is the most effective in generating focused test inputs?*

As detailed in Sections 6.1.2 and 6.1.4, DEEPATASH can be configured with alternative search strategies (single- or multi-objective) and distance metrics (sparseness can be measured on the input, latent or explanation space). This RQ aims at comparing the effectiveness of such six alternative configurations.
RQ2 (Comparison): *How does* DEEPATASH *compare with the state of the art tool DeepHyperion-CS?*

In this RQ, we are interested in whether our focused approach is more effective than *DeepHyperion-CS* in generating test inputs in proximity of and within the target cell. We compare the best performing DEEPATASH configuration (obtained from RQ1) against DEEPHYPERION-CS (introduced in Chapter 5), as the latter is the only state-of-the-art test generator that targets the feature space at large by means of an illumination search algorithm. Unlike Active Learning techniques [114] or unguided test generators, DEEPHYPERION-CS tries to cover all feature combinations and thus it is more likely to produce inputs on the selected target. On the contrary, random techniques produce few or no inputs on the target, making the comparison with DEEPATASH impossible.

Actually, to the best of our knowledge, no state of the art DL test generator is a *focused* test generator, capable of targeting a specific region of the feature space. DEEPHYPERION-CS is a model-based test generator that is applicable to MNIST and IMDB. DEEPHYPERION-CS explores the feature space using the same input representation and mutation genetic operators as DEEPATASH. Therefore, our experimental comparison can effectively rule out all confounding factors and assess the actual contribution of our focused algorithm in isolation.
RQ3 (Usefulness): *Can the test inputs generated by* DEEPATASH *be used to improve the DL system under the test?*

In this RQ, we aim to investigate the usefulness of DEEPATASH in a common DL usage scenario. We simulate a scenario in which a dev2op data shift has been observed, i.e., a feature combination is frequently observed during operation, but it was scarcely (or not) represented at development time. A tester can use DEEPATASH to target the feature values of interest and fine tune the DL system with the generated tests inputs, in order to improve its quality without introducing regressions.

6.2.2 Metrics

We evaluate the focused test generator's effectiveness by measuring the *Tests Close to the target (TC)* as the number of generated failure-inducing inputs in the proximity of the target feature map's cell, i.e. the solutions in the archive whose distances to the target are lower than or equal to 1. Moreover, we assess the generator's ability to reach the target by computing the number of *Tests on Target (TT)*, i.e., the number of failure-inducing inputs that fall within the boundaries of the target cell.

For a given target feature map cell, we prefer a generator that produces diversified inputs. To evaluate this aspect, we measure test input diversity by introducing the *Tests Close to the target Diversity (TCD)* and *Tests on Target Diversity (TTD)* metrics. To this aim, we represent the generated inputs in a lower dimensional space by using the *t-distributed Stochastic Neighbor Embedding (t-SNE)* algorithm [58, 147], which projects similar inputs to neighbouring points and dissimilar inputs to distant points with high probability. Then, we project the inputs generated by all approaches being compared onto the same t-SNE space and compute the clusters of neighbouring points in such a space. The diversity value of each approach is computed as the number of clusters containing at least one input generated by the corresponding approach divided by the total number of clusters [15]: TCD and TTD measure the relative coverage of the clusters by each approach.

We configured t-SNE by choosing 2 as number of dimensions since it performed well in preliminary runs and it eases the results' interpretation. As regards the t-SNE perplexity (which affects the way inputs are scattered or concentrated), we set it to 0.1 after a visual inspection of the plots obtained with different values. For clustering, we applied the *k-Means* algorithm [7] and performed Silhouette analysis [124] to determine the optimal number of clusters k^*, i.e., the value that better balances between cohesion and separation of the clusters.

Figure 6.3 exemplifies the diversity comparison between three DEEPATASH configurations. The points represent the inputs generated by each configuration in the 2D t-SNE space. Each configuration is assigned a different color. Points are grouped into clusters (represented as circles) and diversity is computed as the number of clusters covered by each configuration. In this example, the diversity value for NSGAII-Input is 0.1; it is 0.5 for NSGAII-Explanation and 0.7 for NSGAII-Latent.

6.2.3 Evaluation Scenarios

A crucial aspect of focused input generation is the choice of the target cells. Developers can use information from the operation environment to identify feature map cells that

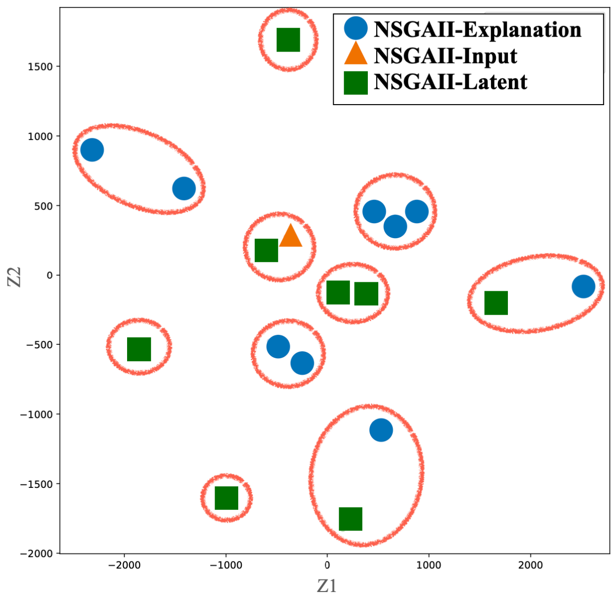


Figure 6.3. Example t-SNE plot to explain the computation of test diversity metrics, with clusters represented as empty circles containing inputs (smaller, solid shapes).

occur in operation, but are under-represented in the train/test set. Since we do not have access to operation data, we chose our targets starting from the *misbehaviour probability map* (such as the one in Figure 6.1), a feature map that encodes the DL failure probability for different feature combinations. In the misbehaviour probability map shown in Figure 6.1, the darkness level of the cells is proportional to their Average Misbehaviour Probability (AMP) values; thick borders highlight combinations producing misbehaviours with high confidence, while blank cells correspond to uncovered feature combination values. We leveraged misbehaviour probability maps and AMP to mimick various possible user choices by evaluating DeepAtash in the following scenarios:

Dark Targets: targets selected among the dark, thick-bordered cells (misbehaviour probability > 0.8 and confidence interval > 0.65). These cells correspond to error-prone feature values and mostly contain individuals with high probability of causing misbehaviours. In this scenario, the user wants to collect diverse new inputs with critical characteristics, e.g., to fine tune the DL system;

Grey Targets: targets corresponding to covered cells with misbehaviour probability ≤ 0.8. These cells correspond to feature value combinations for which the DL system generally behaves correctly. Therefore, in this scenario the user wants to stress the DL system with inputs whose characteristics seem not critical, to see if they can possibly trigger any misbehaviour;

White Targets: targets corresponding to uncovered cells, i.e., cells that do not contain inputs. In this scenario, the user wants to generate new data with missing

Table 6.1. Hyperparameters used in the experiments

Parameter	MNIST	IMDB
seed pool size	800	1000
population size	100	100
time budget (s)	3600	3600
mutation lower bound	0.01	-
mutation upper bound	0.6	-
sentimentDist	-	5
maxDist	-	5
repopulation upper bound	10	10
target archive size	81	42
number of epochs for retraining	6	5
learning rate for retraining	0.001	0.0001

feature combinations or check the feasibility of the selected combination of feature values.

Since the usage scenario we are mimicking is one where operation data occur in regions of the feature map that are under-populated, for dark and grey cells we apply the additional filter that the selected cell must contain a number of individuals lower than the average number of individuals observed across all feature map cells (this filter is not necessary for white cells, which are not populated at all).

6.2.4 Experimental Procedure

To answer our research questions, we ran DEEPATASH in the three evaluation scenarios introduced in Section 6.2.3 along with DEEPHYPERION-CS on the subject systems, in all the possible 2D combinations of the proposed features. For each scenario, the first step is the selection of the target cell from the misbehaviour probability maps generated by DEEPHYPERION-CS.

As regards the dark and grey targets, we chose a cell among the underpopulated dark and grey cells of the misbehaviour probability map. More specifically, we randomly chose a cell for which the coverage (i.e., number of individuals assigned to the cell) achieved by DEEPHYPERION-CS is lower than the average cell coverage, computed by considering all cells in all DEEPHYPERION-CS runs. As white targets, we chose uncovered cells in the DEEPHYPERION-CS misbehaviour probability maps. Since uncovered cells may be unfeasible, we considered white cells in the neighborhood of covered cells.

Then, we ran DEEPATASH focusing on the identified target cells and collected the resulting archives of solutions.

For our experiments, we used three features defined for MNIST digits (see Section 5.1.3): (1) *Luminosity (Lum)*: number of light pixels of the image, i.e., pixels whose value is above 127; (2) *Orientation (Or)*: vertical orientation of the digit, obtained by computing the angular coefficient of the linear regression of the non-black pixels; (3) *Moves* (Mov): sum of the Euclidean distances between pairs of consecutive sections of the digit.

We used three features defined for IMDB movie reviews: (1) *Positive words count (Pos)*: number of words in the text with positive polarity, obtained by counting the words tagged as positive in the English Opinion Lexicon by Liu and Hu [59]; (2) *Negative words count (Neg)*: number of words in the text with negative polarity, obtained by counting the words tagged as negative in the aforementioned lexicon; (3) *Verb count (Verb)*: number of verbs in the text, a proxy for the text complexity, computed by counting the words with a *verb* tag, according to the part-of-speech tagging performed by the NLTK library[2].

The hyperparameters of DeepAtash were obtained whenever possible from the experiments conducted with DeepHyperion-CS and were fine tuned in a few preliminary runs to ensure the target cells are reachable with them. The resulting hyperparameter values are reported in Table 6.1. For DeepHyperion-CS, we used the latest version of the tool along with the configuration reported in Chapter 5.

To facilitate a fair comparison, we used the same initial seeds for the compared tools. The seeds for MNIST were obtained by considering all test set inputs that belong to class “5”. For IMDB, the seeds were selected from the test set inputs, all belonging to class “positive”. Since we had a huge number (i.e., 12500) of positive reviews in the IMDB test set, we randomly selected an initial seed pool of 1000 inputs among the ones closer to the target (with distance lower than a threshold *maxDist*; see Table 6.1).

To allow statistical analysis, we ran each approach 10 times for each target, for a total of 240 runs on each subject. To ensure a fair comparison, we ran all tools with the same time budget (1h for MNIST and IMDB). To assess the statistical significance of the comparisons between different DeepAtash configurations (RQ1), and between DeepAtash and DeepHyperion-CS (RQ2), we applied the Mann-Whitney U-test and measured the effect size by means of the Vargha-Delaney’s $\hat{A}_{12}$ statistic [5].

To answer RQ3, we fine tuned [12] the original DL models and trained them for additional epochs at the same or lower learning rate (see Table 6.1) with the inputs generated by DeepAtash close (TC) and within (TT) the considered targets. First, for each feature combination, we collected the inputs generated by DeepAtash within and close to the targets. Then, we equally divided these inputs into two sets, i.e., $training_{DA}$ and $test_{DA}$. The combination of the original training set and $training_{DA}$ was used to fine tune the DL system, while the original test set and $test_{DA}$ were used to evaluate the accuracy of the fine-tuned DL system. We repeated the fine tuning procedure 10 times for each run of DeepAtash on each target cell, to measure the statistical significance of the accuracy improvement.

[2]Natural Language Toolkit - `https://www.nltk.org`

6.2.5 Results

RQ1: Effectiveness

Table 6.2 reports the results achieved on MNIST and IMDB by the 6 alternative configurations of DeepAtash, obtained by combining the search strategies (GA and NSGA-II) and the sparseness metrics (i.e., distance computed in the Input, Latent and Explanation space).

For each evaluation scenario described in Section 6.2.3, the table reports a row for each feature combination. The columns report the number of failure-inducing inputs generated in the proximity of the target (TC), those of them exactly on target (TT), and their diversity (TCD and TTD, respectively). For each target-feature combination (row), the largest value is highlighted in bold, while the underlined values are comparable among them (p-value ≥ 0.05) and significantly higher than the remaining ones (p-value < 0.05, large or medium effect size), which, when they exist, are not underlined.

As regards MNIST (Table 6.2 - top), the configuration NSGA-II + Latent produced significantly more and sparser inputs close to the target (TC and TCD) than the other configurations, on average across all the target-feature combinations (39% of them on target). Instead, in terms of TT and TTD, NSGA-II + Latent performed comparably to GA + Latent and GA + Input (p-value > 0.05), and significantly better than the other DeepAtash configurations.

For IMDB (Table 6.2 - bottom), NSGA-II + Latent performed significantly better than the other configurations, on average across all the target-feature combinations and for all the considered metrics.

We can observe that in terms of sparseness (metrics TCD, TTD), heatmaps (column "Explanations") perform consistently worse across most configurations, with only a few exceptions. This may be either due to the information that they discard by considering only input elements that contribute to a prediction, or to the heatmap computation itself, which introduces an overhead and hence consumes the overall test generation budget more quickly.

RQ1: *Overall, the multi-objective* DeepAtash *configuration guided by the sparseness metric computed in the latent space generated a significantly larger number of diverse inputs close to the target than the other tool configurations. For* IMDB, *this configuration performs significantly better also in terms of tests on target.*

Table 6.2. RQ1 - Tests close to target (TC), tests on target (TT), tests close to target diversity (TCD), and tests on target diversity (TTD) by alternative DeepAtash configurations for MNIST and IMDB. In each row, boldface indicates the maximum; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

			Input				Latent				Explanation			
			GA		NSGA-II		GA		NSGA-II		GA		NSGA-II	
		Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
MNIST	Dark	Mov-Lum	41.10 [**0.38**]	41.10 [0.48]	43.10 [0.36]	23.90 [0.12]	**51.70** [0.37]	**51.70** [0.47]	50.50 [0.31]	46.80 [**0.58**]	16.40 [0.24]	1.00 [0.05]	33.60 [0.28]	1.80 [0.25]
		Mov-Or	**69.10** [0.28]	4.40 [0.10]	58.60 [0.22]	1.60 [0.15]	66.10 [**0.30**]	**10.20** [**0.45**]	55.30 [0.22]	1.20 [0.13]	28.50 [0.14]	0.00 [0.00]	26.70 [0.12]	3.00 [0.25]
		Or-Lum	**70.30** [0.27]	24.20 [0.65]	32.10 [0.14]	5.90 [0.35]	65.50 [**0.31**]	**25.90** [**0.70**]	64.30 [0.22]	15.60 [0.50]	55.30 [0.21]	8.60 [0.55]	42.80 [0.18]	5.80 [0.45]
	Grey	Mov-Lum	41.40 [0.60]	**38.30** [**0.73**]	21.80 [0.28]	0.00 [0.00]	22.10 [0.38]	16.20 [0.35]	**53.40** [**0.78**]	28.40 [0.40]	5.10 [0.25]	0.60 [0.08]	13.40 [0.49]	0.40 [0.10]
		Mov-Or	18.50 [0.45]	3.00 [0.20]	15.30 [0.43]	2.40 [0.20]	16.00 [0.24]	0.70 [0.11]	**20.50** [**0.53**]	**4.70** [**0.28**]	13.10 [0.30]	1.80 [0.15]	14.90 [0.49]	0.40 [0.10]
		Or-Lum	10.10 [0.29]	**10.10** [0.34]	22.90 [0.47]	8.60 [0.27]	9.20 [0.23]	9.20 [0.30]	19.20 [0.52]	6.80 [0.19]	6.40 [0.29]	3.50 [0.24]	**28.10** [**0.56**]	6.20 [**0.55**]
	White	Mov-Lum	14.30 [0.36]	**11.60** [0.28]	28.20 [0.61]	2.30 [0.30]	20.60 [0.42]	10.70 [**0.36**]	**29.60** [0.54]	10.40 [0.25]	10.90 [0.38]	5.10 [0.15]	15.40 [0.60]	6.20 [0.55]
		Mov-Or	24.60 [0.44]	2.00 [0.21]	11.30 [0.31]	**7.70** [0.10]	22.10 [0.50]	5.90 [**0.35**]	**25.10** [**0.65**]	1.80 [0.18]	6.70 [0.43]	0.00 [0.00]	7.30 [0.32]	0.00 [0.00]
		Or-Lum	23.30 [0.48]	21.60 [0.58]	30.20 [0.52]	10.10 [0.38]	28.70 [0.51]	24.30 [**0.71**]	**51.00** [**0.66**]	**28.00** [0.65]	21.50 [0.51]	6.80 [0.48]	20.80 [0.52]	4.30 [0.48]
		AVG	34.74 [0.39]	**17.37** [0.40]	29.28 [0.37]	6.94 [0.21]	33.56 [0.36]	17.20 [**0.42**]	**40.99** [**0.49**]	15.97 [0.35]	18.21 [0.31]	3.00 [0.19]	22.56 [0.40]	2.68 [0.26]
IMDB	Dark	Neg-Pos	33.00 [0.68]	22.00 [0.5]	29.40 [0.53]	6.90 [0.34]	25.80 [0.58]	14.30 [0.53]	40.30 [0.74]	33.40 [0.64]	25.70 [0.49]	7.8 [0.31]	29.90 [0.51]	9.60 [0.37]
		Neg-Verb	7.20 [0.31]	3.20 [0.12]	9.20 [0.36]	6.30 [0.16]	5.00 [0.16]	0.70 [0.10]	27.10 [0.63]	14.60 [0.34]	11.20 [0.53]	3.60 [0.15]	19.70 [0.54]	6.60 [0.17]
		Pos-Verb	33.80 [0.53]	33.80 [0.43]	31.60 [0.45]	31.60 [0.47]	33.70 [0.50]	33.40 [0.58]	37.60 [0.52]	37.60 [0.57]	28.80 [0.50]	27.80 [0.38]	28.30 [0.47]	6.60 [0.17]
	Grey	Neg-Pos	**7.50** [0.30]	5.20 [0.35]	6.40 [0.22]	3.50 [0.09]	5.00 [0.25]	3.80 [0.06]	**10.80** [**0.52**]	**8.30** [**0.35**]	0.80 [0.05]	0.00 [0.00]	5.60 [0.34]	0.10 [0.00]
		Neg-Verb	7.30 [0.45]	7.30 [0.41]	15.00 [0.57]	**14.90** [0.61]	10.70 [0.51]	10.70 [0.49]	**15.70** [**0.63**]	13.10 [**0.62**]	9.50 [0.54]	8.60 [0.54]	12.20 [0.45]	11.20 [0.55]
		Pos-Verb	27.10 [0.50]	27.10 [0.60]	28.70 [0.49]	28.70 [0.49]	24.10 [0.62]	27.30 [0.41]	**32.00** [0.53]	**32.00** [0.56]	27.30 [0.41]	27.30 [**0.68**]	25.50 [0.62]	25.50 [0.58]
	White	Neg-Pos	24.20 [0.65]	15.40 [**0.65**]	31.10 [0.62]	22.40 [0.63]	29.40 [0.64]	18.00 [0.53]	**38.90** [0.60]	**29.20** [0.63]	24.30 [0.52]	20.90 [0.58]	26.40 [**0.67**]	14.20 [0.58]
		Neg-Verb	4.10 [0.26]	1.80 [0.10]	0.00 [0.00]	0.00 [0.00]	5.60 [**0.52**]	0.00 [0.00]	**13.00** [0.37]	**3.60** [**0.25**]	0.70 [0.06]	0.00 [0.00]	0.40 [0.02]	0.00 [0.00]
		Pos-Verb	6.10 [0.13]	2.30 [0.10]	25.10 [**0.60**]	**9.40** [0.38]	3.00 [0.10]	1.70 [0.07]	**25.70** [0.48]	3.80 [0.19]	8.40 [0.15]	0.00 [0.00]	15.90 [0.53]	1.30 [0.07]
		AVG	16.70 [0.42]	13.12 [0.36]	19.60 [0.43]	13.70 [0.35]	15.81 [0.43]	11.86 [0.33]	**26.80** [**0.56**]	**19.51** [**0.46**]	19.60 [0.43]	13.70 [0.35]	18.21 [0.46]	10.80 [0.30]

RQ2: Comparison

Table 6.3 compares the results achieved on MNIST and IMDB by the best DeepAtash configuration (i.e., NSGA-II - Latent) with the baseline tool DeepHyperion-CS.

For MNIST and IMDB, DeepAtash achieved significantly larger TC, TCD, TT and TTD values than DeepHyperion-CS, on average across all the target-feature combinations. DeepAtash generated up to 29.2 more inputs on targets than DeepHyperion-CS. In white cells where the competitor generated none (e.g., for IMDB, White target, Neg-Pos feature combination), DeepAtash was able to find some inputs with the desired feature combinations. DeepAtash outperformed DeepHyperion-CS for all the target-feature combinations (with statistical significance 80% of the times).

These results confirm that our focused approach can generate inputs in feature space areas where state-of-the-art, general-purpose generator DeepHyperion-CS can generate few or no inputs.

RQ2: DeepAtash *outperforms the state of the art tool* DeepHyperion-CS *in generating misbehaviour-inducing inputs with target feature value combinations.*

RQ3: Usefulness

Table 6.4 shows the accuracy improvement achieved by fine tuning the considered DL systems with $training_{DA}$, the training partition of the inputs generated by DeepAtash. The "before" columns show the accuracy of the original DL systems on the original test set and $test_{DA}$, the test set partition generated by DeepAtash. The "after" columns show the accuracy values achieved after fine tuning the DL systems with $training_{DA}$. All values in the "after" columns are underlined, which indicates statistically significant accuracy improvement after the fine tuning (p-value < 0.05, large effect size).

Since we selected state-of-the-art DL systems, their initial accuracy on the original test set was quite high, i.e., 99.11% for MNIST and 88.19% for IMDB. On the other hand, their initial accuracy on $test_{DA}$ was obviously 0% since we considered misbehaviour-inducing inputs generated by DeepAtash.

Quite surprisingly, by fine tuning the considered DL systems using $training_{DA}$, we improved their accuracy on the original test set despite their initial high quality. In fact, for all feature combinations, the accuracy on the original test set significantly increased (up to 1.39% for Neg-Pos). This might be due to an increased generalization capability induced by the additional training on inputs with under-represented feature combinations. So, not only we observed no sign of regressions, but we also achieved a slight accuracy improvement on the original test set. As expected, the accuracy on $test_{DA}$ dramatically increased from 0% to at least 97.35%.

Table 6.3. RQ2 - Results achieved by the compared tools for MNIST and IMDB. Tests close to target (TC) and their diversity (TCD); tests on target (TT) and their diversity (TTD). In each row, boldface is the maximum; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

			DeepAtash		DeepHyperion-CS	
		Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
MNIST	Dark	Mov-Lum	**50.50** [**0.90**]	**46.80** [**0.97**]	18.30 [0.38]	2.30 [0.07]
		Mov-Or	**55.30** [**0.86**]	1.20 [0.27]	37.60 [0.47]	**2.40** [**0.42**]
		Or-Lum	**64.30** [**0.95**]	**15.60** [**0.74**]	13.80 [0.30]	2.70 [0.15]
	Grey	Mov-Lum	**53.40** [**0.81**]	**28.40** [**0.50**]	22.70 [0.50]	3.70 [0.10]
		Mov-Or	20.50 [**0.88**]	**4.70** [**0.45**]	**24.70** [0.45]	1.10 [0.15]
		Or-Lum	**19.20** [**0.81**]	**6.80** [**0.39**]	2.20 [0.19]	0.10 [0.01]
	White	Mov-Lum	**29.60** [**0.74**]	**10.40** [**0.40**]	11.90 [0.42]	0.00 [0.00]
		Mov-Or	**25.10** [**0.71**]	**1.70** [**0.20**]	20.70 [0.50]	0.00 [0.00]
		Or-Lum	**51.00** [**1.00**]	**28.00**[**1.00**]	0.80 [0.05]	0.00 [0.00]
	AVG		**40.99** [**0.85**]	**15.96** [**0.55**]	16.97 [0.36]	1.37 [0.10]
IMDB	Dark	Neg-Pos	**40.30** [**0.94**]	**33.40** [**1.00**]	8.20 [0.11]	1.60 [0.05]
		Neg-Verb	**27.10** [**1.00**]	**14.60** [**0.43**]	10.80 [0.05]	4.50 [0.07]
		Pos-Verb	**32.00** [**0.95**]	**32.00** [**1.00**]	2.40 [0.05]	1.10 [0.05]
	Grey	Neg-Pos	**10.80** [**0.73**]	**8.30** [**0.40**]	10.20 [0.20]	1.00 [0.20]
		Neg-Verb	**15.70** [**0.95**]	**13.10**[**0.93**]	7.70 [0.11]	1.80 [0.08]
		Pos-Verb	**37.60** [**0.95**]	**37.60** [**0.95**]	12.00 [0.15]	5.20 [0.11]
	White	Neg-Pos	**38.90** [**1.00**]	**29.20** [**1.00**]	0.20 [0.00]	0.00 [0.00]
		Neg-Verb	**13.00** [**0.50**]	**3.60** [**0.30**]	0.30 [0.10]	0.00 [0.00]
		Pos-Verb	**25.70** [**0.70**]	**3.50** [**0.30**]	0.70 [0.10]	0.00 [0.00]
	AVG		**26.79** [**0.86**]	**19.48** [**0.70**]	5.83 [0.10]	1.69 [0.06]

Table 6.4. RQ3 - Model Accuracy (ACC) on the original test set and on the test set generated by DeepAtash, before and after fine tuning the DL system with the training partition of generated inputs. In each row, boldface indicates the maximum; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

		Original Test Set		DA Test Set	
	Features	ACC before	ACC after	ACC before	ACC after
MNIST	Mov-Lum	99.11	**99.23**	0.00	**99.92**
	Mov-Or		**99.24**		**99.65**
	Or-Lum		**99.23**		**99.02**
IMDB	Neg-Pos	88.19	**89.58**	0.00	**98.36**
	Neg-Verb		**89.56**		**99.47**
	Pos-Verb		**89.56**		**97.35**

RQ3: DEEPATASH *is useful to improve the accuracy of a DL system trough fine tuning, by targeting feature combinations under-represented or unseen during development.*

Threats to Validity

External Validity: The choice of subjects might have threatened the external validity. We chose DL systems widely used in SE research that belong to separate domains. In particular, we chose subjects for which semantic information on the inputs is accessible. In fact, the key requirement for DEEPATASH is that a generative model of the inputs exists, such that genetic operators can operate on the generative model's parameters. Therefore, our main limitation is the availability of a generative input model. Generative models are widely used in many domains, such as cyber-physical systems, where the environment is often modeled and simulated. In domains where a model would be prohibitively expensive, like image processing, generative neural networks (e.g., GANs [44, 28]) can be used as an input model. Replication of our experiments on additional case studies would be important to corroborate our findings. Another external validity threat is introduced by the choice of the targets, because results may not generalize to different choices of the target. To mitigate this threat, we chose three types of targets (Dark, Grey, White), corresponding to different usage scenarios.

Conclusion Validity: The stochastic nature of DL systems and search-based approaches may affect the results. Therefore, we ran each experiment multiple times and conducted standard statistical tests to assess the results' significance.

6.3 Focused Test Generation for Autonomous Driving Systems

Autonomous Driving Systems (ADSs) have become popular due to their potential to revolutionize transportation by enhancing safety, improving efficiency, and providing a more convenient and accessible mode of travel for individuals around the world. On the other hand, the consequences of a misbehaviour might be catastrophic for life-critical systems like ADSs, potentially endangering all road participants (e.g., driver, passengers, and pedestrians). For instance, a lane-keeping assist system (LKAS) might predict an erroneous steering angle when presented with a road with an exceptionally sharp turn, if this particular variation is not adequately represented in the training set. Hence, ADSs necessitate rigorous testing employing appropriate techniques capable of generating data beyond the datasets used during development [118, 140].

Through its search-based focused testing approach, DeepAtash proved its effectiveness by generating multiple diverse inputs with predefined target feature values for different DL systems, i.e., image and text classifiers. However, DeepAtash may not be the ideal choice for testing ADSs. In fact, its evolutionary nature demands multiple system executions to produce misbehaviour-inducing inputs within the specified target. Testing ADSs is known to be resource-intensive, since it requires expensive executions (e.g., simulations) to assess the system behaviour. Consequently, DeepAtash might invest most of its test generation time-budget exploring unpromising regions of the feature space, i.e., those that do not contain the target features or have a minimal likelihood of causing misbehaviours, because each execution (i.e., simulation in the candidate test scenario) consumes a substantial fraction of the available budget.

We propose a novel focused test generator named DeepAtash-LR, specifically designed for ADSs, with the primary goal of overcoming the aforementioned limitations of its predecessor. The core innovation of DeepAtash-LR is its ability to reduce the computational resource demands associated with the evaluation of less promising solutions. Our approach achieves this goal by integrating a surrogate model within the evolutionary process. Surrogate models have been extensively used for ADS testing since they closely emulate system behaviour, while drastically reducing computational overhead [104]. Within DeepAtash-LR, the surrogate model plays a pivotal role in predicting both the likelihood of a misbehaviour and the behavioural features of the generated inputs. Consequently, DeepAtash-LR executes the system only when the input is likely to belong to the target feature map cell and is likely to trigger a misbehaviour. This strategic approach optimizes the utilisation of computational resources in ADS testing and makes a dramatically better use of the available test generation budget.

Our evaluation of DeepAtash-LR was conducted in the context of a LKAS DL component, using the BeamNG driving simulator [10] across different usage scenarios. Our empirical results show that the surrogate model is indispensable for generating misbehaviour-inducing inputs within the predefined targets. Moreover, the inputs generated by DeepAtash-LR have been used to fine tune the ADS under study and enhance its performance on under-represented feature combinations, which were initially not handled at all (i.e., on failure-inducing feature combinations). After fine tuning, the system improved remarkably, with no significant regressions.

6.4 Focused Testing with Surrogate Models: the DeepAtash-LR Technique

The original DEEPATASH algorithm (Section 6.1) computes the closeness to misbehaviour of the generated inputs by running the ADS in the generated driving scenarios within a simulator, which can be quite computationally expensive. It should be noticed that also the computation of behavioural features requires the execution of the ADS under test in a simulator. This translates into multiple executions of the ADS under test in each iteration. Even though evolutionary search algorithms generally demonstrate good scalability, their ability to effectively generate inputs with specific features may decrease when the evaluation of such inputs is expensive. A clear example of this occurs when testing the lane-keeping assist component of an ADS. In this context, the assessment of the system's behaviour in each input scenario requires the execution of time-intensive simulations, which eventually constrain the evolutionary process by restricting the number of iterations performed within a given time budget (i.e., the number of generations of the evolutionary search). We address this limitation by proposing a way to improve the efficiency of the search-based algorithm, which translates also into its effectiveness, as a better use of the available time budget might lead to higher fault detection. Specifically, we employ surrogate models to predict the system's behaviour without the need for executing time-consuming simulations and, thus, saving time during the optimization process. Surrogate models have recently gained popularity [1, 48, 15, 52, 104] for emulating the behaviour of the original system. Surrogate models are trained on datasets containing inputs and the corresponding behaviours of the systems. They approximate and mimic the behaviour of the original system without executing it, hence providing a faster and more efficient way to evaluate the fitness of candidate test inputs, although such evaluation is affected by some degree of approximation, due to the use of a surrogate instead of the real system. Hence, their practical usefulness depends on the degree of approximation involved and on the tolerance to approximation errors of the test generation algorithm, which can be only assessed empirically. With surrogate models, ADS simulations are executed only when necessary, such as when collecting data to train the surrogate model or when storing an input in the archive as a final solution. The surrogate model allows the algorithm to decide whether an input is close enough to the target or exhibits the desired characteristics without the need to run costly simulations for all inputs.

Algorithm 4 presents an overview of our technique. This general pseudocode can be instantiated either as a single- or multi-objective optimization process. The highlighted lines of pseudocode indicate the changes over the original DEEPATASH algorithm. The

Algorithm 4: DeepAtash-LR's Focused Test Generation

```
Input  : $B$: execution budget
         $archivesize$: target archive size
         $S$: set of input seeds
         $popsize$: population size
         $targetCell$: target of focused test generation
         $epsilon$: threshold for surrogate training
Output : $A$: archive of test inputs in the target cell
1  $A \leftarrow \emptyset$;
2  population $P \leftarrow$ InitialisePopulation($S$, $popsize$);
3  $UseSurrogate \leftarrow False$ ;
4  Evaluate($P$, $A$, $targetCell$, $UseSurrogate$) ;
5  $TrainingDS \leftarrow$ P ;
6  while $elapsedBudget < B$ do
7      if $elapsedBudget > epsilon \times B$ and $UseSurrogate$ is $False$ then
8          TrainSurrogate($TrainingDS$) ;
9          $UseSurrogate \leftarrow True$ ;
10     end
11     offspring $Q \leftarrow P$ ;
12     foreach $q \in Q$ do
13         $q \leftarrow$ Mutate($q$) ;
14     end
       // substitute the worst individuals
15     $P \leftarrow$ Repopulation($P$, $S$, $A$);
16     Evaluate( $P \cup Q$, $A$, $targetCell$, $UseSurrogate$) ;
17     if $UseSurrogate$ is $False$ then
18         $TrainingDS \leftarrow TrainingDS \cup Q$ ;
19     end
20     $A \leftarrow$ UpdateArchive($P \cup Q$, $archiveSize$, $targetCell$, $UseSurrogate$);
21     $P \leftarrow$ Select($P \cup Q$, $popsize$);
22 end
23 $A \leftarrow$ FilterMisbehaviours($A$);
24 return $A$
```

algorithm starts by initialising an empty archive *A* (line 1), that will store the best test inputs generated during the search, i.e., the most sparse inputs with feature values falling within or close to the target ranges. Then, the algorithm proceeds by generating an initial population *P* (function InitialisePopulation at line 2), consisting of a specified number of individuals. These individuals are instantiated by selecting elements from an initial pool of seeds *S*, which is provided as input to the algorithm. Typically, *S* can be a subset of the test set available in the data set of the ADS under test. Indeed, if a test set is not available or desirable, the subset *S* can alternatively be composed of randomly generated inputs to serve as the initial pool for the test input generation process. The first phase is concluded by determining the fitness values of all the individuals of the initial population without using the surrogate model (lines 3-4). Since there is no data available about the behaviour of the original ADS at this stage, we need to prepare the training data set for the surrogate model based on these initial evaluations. Then, the algorithm initialises *TrainingDS*, i.e., the training data set for the surrogate model, with the inputs from the current population *P* (line 5).

The main evolutionary loop is performed until the termination condition is met (lines 6-22). During the evolutionary loop, training of the surrogate model occurs only when a sufficient number of iterations has been performed and, thus, enough inputs have been generated to train the surrogate model. Once this condition is satisfied (line 7), the algorithm trains the surrogate model (line 8), and sets the *UseSurrogate* variable to *True* (line 9). At each iteration, the population is mutated by genetic operators to produce its offspring *Q* (lines 11-14). The Repopulation operator avoids stagnation in local optima by replacing the worst individuals of the population (i.e., the ones with the lowest fitness values) with new inputs selected from the initial pool of seeds *S* (line 15). In particular, Repopulation takes as input the archive *A* to avoid selecting seeds already used to generate individuals present in the current archive *A*.

Function Evaluate calculates the fitness of the current population *P* and its offspring *Q*. This function avoids redundant evaluations of the distance from the target cell and the closeness to misbehaviour for the individuals that have been already evaluated in the previous iterations, i.e., the individuals from P that have not undergone repopulation. However, the evaluation of sparseness is performed at each iteration as it measures how dissimilar is each individual from solutions previously discovered and currently stored in the archive. In fact, the sparseness value of an individual may vary at each iteration based on the current composition of the archive. The evaluation of the behaviour and the feature values can be performed either using the surrogate model or the system under test depending on the value of the *UseSurrogate* variable (line 16). The evaluated inputs are added to the training data set of the surrogate model only when we have used the actual ADS for their evaluations (line 17-19).

The inputs that are close to the target cell, i.e., those with a distance from the target cell smaller than a certain threshold, are saved in the archive *A* if they outperform the previously found solutions. In other words, if these inputs exhibit better fitness than the existing stored solutions, they are added to the archive (line 20). It should be noticed that when an archive replacement is supposed to happen, because a better individual was generated, the actual fitness values are needed. If the individual's evaluation was approximated by the surrogate model, when the individual becomes a candidate for the archive it must be first evaluated against the real system, which means the ADS must be simulated in the candidate input scenario.

Then, the SELECT function selects the *popsize* fittest individuals for the next generation (line 21). When dealing with the optimization of multiple fitness functions simultaneously, the ranking of individuals for selection in DEEPATASH-LR is determined based on the principles of Pareto dominance and crowding distance, as prescribed by the NSGA-II multi-objective optimization algorithm [23] (see Section 6.1.4). When the execution budget *B* is finished, the algorithm filters the misbehaviour-inducing inputs from the archive and reports them as final outcome (lines 23-24), together with the AMP values of each feature map cell.

In the rest of this section, we will describe in detail only the key aspects of DEEPATASH-LR that differ from DEEPATASH, and how we have applied it to the lane-keeping application domain.

6.4.1 Input Representation

DEEPATASH-LR can be classified as a model-based input generation technique [146], since it represents individuals, i.e., test inputs, as model instances and directly manipulates the model to generate new data.

In our setting, test inputs are scenarios in which the car drives within the BEAMNG simulator. Following the state-of-the-art [40, 119, 159, 161], we represent a driving scenario to test the lane keeping component as a plain asphalt road surrounded by green grass on which the car has to drive by keeping the right lane. Such simulated roads are modelled as a sequence of consecutive points in a bi-dimensional space, interpolated by Catmull-Rom cubic splines [21] (for more details see Section 5.2.1).

6.4.2 Fitness Functions

DEEPATASH-LR's optimization process utilises three fitness functions. They quantify: (1) the distance of the test input from the target cell (see Section 6.1.2); (2) the closeness of the ADS to cause a misbehaviour when executing the given test input; and, (3) the distance of the input from the already found solutions (i.e., its sparseness).

Closeness to Misbehaviour

DEEPATASH-LR aims to generate test inputs that trigger misbehaviours of the ADS under test. Therefore, it employs a problem-specific fitness function that quantifies how close the ADS is to misbehaving when exercised with the evaluated input. By minimizing this fitness function, DEEPATASH-LR identifies inputs that are more likely to trigger misbehaviours (see Section 6.4).

For the *lane-keeping problem*, we characterise the behaviour of the system by the maximum distance of the car from the center of the right lane during the simulation [137, 63, 119]. The fitness value is calculated as $min(w/2-d)$, where w is the lane width and d the distance of the car from the centre of the right lane. This fitness function gets its maximum value ($w/2$) when the car is at the center of the right lane, whereas it gets a negative value when there is a misbehaviour, i.e., the car is beyond one of the lane margins. All instances where this fitness function has a negative value indicate a misbehaviour, i.e., the vehicle does not keep the lane. To ensure equal importance for all misbehaviours, we cap the value of this fitness function for all misbehaviour-inducing individuals to a small negative value (i.e., -0.1). To evaluate this fitness function, DEEPATASH-LR can use either the original ADS under test or the surrogate model trained with the inputs generated during the evolution, along with the corresponding behaviours observed through simulation.

Sparseness

Ensuring diversity and distinctiveness in the generated test inputs is essential for the effectiveness of a focused test input generator. By generating a wide variety of inputs that are significantly different from one another, the generator can cover a broader range of execution scenarios, which helps in thoroughly exploring the system's behaviour.

To achieve this objective, DEEPATASH-LR uses a fitness function that quantifies how dissimilar an input is from the solutions previously discovered during the search. By maximizing this fitness function, DEEPATASH-LR encourages the generation of novel and unique inputs, leading to a richer and more comprehensive set of test cases that can reveal different aspects of the system's behaviour (see Section 6.5).

Specifically, we use the weighted Levenshtein distance [119]. This metric considers the minimum number of edit operations to transform one road into the other. Edit operations apply to the two sequences of angles sampled on the spines of the two roads being compared. This distance metric is suitable for the comparison of shapes of roads with different lengths and is robust against translation and rotation. In fact, it takes into account the order of the points along the road spines, as well as the relative angle between consecutive points.

6.4.3 Surrogate Model

As shown in Algorithm 4, each iteration of DEEPATASH-LR involves the evaluation of newly generated individuals to compute their behavioural features and fitness values. This evaluation can be performed using either the original ADS or the surrogate model, depending on the value of the *UseSurrogate* variable. The surrogate model functions as a black-box component, accepting test input (e.g. sequence of control points which represents the road shape) as input and generating the desired variable (e.g. a behavioural feature) as output. DEEPATASH-LR requires a set of labeled inputs to train an accurate surrogate model. Consequently, a portion of the time budget is dedicated to evaluating the generated inputs using the original ADS under test through simulations. Once a reasonable amount of training data has been accumulated (i.e., after a given number of iterations), this data is utilized to train the surrogate model. The amount of collected training data is determined using the threshold ϵ in Algorithm 4, which defines the portion of time budget dedicated to training data collection. This threshold can be chosen through preliminary runs to ensure that the performance of the surrogate model is satisfactory. It is essential to set the threshold ϵ at a relatively low value; otherwise, adopting the surrogate model would be inefficient as the training data collection time and the training time would be excessively long.

The trained surrogate model becomes a valuable asset for DEEPATASH-LR within the remaining time budget. Instead of relying on the original ADS for evaluations, our tool can use the surrogate model to predict the behaviour of the ADS for new inputs (line 9). This replacement may reduce the time and computational resources needed for evaluations, if the surrogate model is capable of delivering fast and accurate predictions. For problems such as lane-keeping, where input evaluation involves costly simulations, utilizing the surrogate model can significantly impact the exploration of the algorithm.

We adopted three distinct surrogate models to predict two behavioural features (i.e. mean lateral position and standard deviation of steering angle) along with the closeness to misbehaviour (i.e. distance to the road boundaries). These predictions would need the execution of simulations in the original configuration of DEEPATASH.

6.4.4 Archive of Solutions

The UPDATEARCHIVE function that manages the fixed-size archive of solutions is described in Algorithm 5. In DEEPATASH-LR, the surrogate model can be employed to predict the behaviour of the original ADS, allowing for faster fitness evaluations during the evolutionary process. However, when an individual is deemed as eligible for inclusion in the archive by the surrogate model, i.e., its predicted distance from the target cell is lower or equal to 1, DEEPATASH-LR re-evaluates it by performing a simulation using the original ADS. As shown in lines 3-4 of Algorithm 5, the re-evaluation takes place only if the candidate was evaluated through the surrogate model, instead of performing a simulation. Such re-evaluation recomputes the values of the behavioural features and of the fitness functions, allowing to obtain a more accurate and reliable assessment. In fact, this additional evaluation step ensures that individuals that are close enough to the target cell undergo a final assessment with the original ADS to confirm their suitability for inclusion in the archive. By doing so, DEEPATASH-LR maintains the integrity of the archived solutions and ensures that the best-performing inputs are validated against the real ADS system.

Algorithm 5: The UPDATEARCHIVE function

```
Input  : P: population
         A: initial archive
         targetCell: target feature value ranges
         UseSurrogate: if surrogate model is used or not
Output : A: updated archive
1  foreach $p \in P$ do
2      if DIST(p, targetCell) $\leq$ 1 then
3          if UseSurrogate then
4              EVALUATE(p, A, targetCell, UseSurrogate=False);
5          end
6          if A is not full and $p \notin A$ and DIST(p, targetCell) $\leq$ 1 then
7              A.insert(p) ;
8          else
9              ind $\leftarrow$ GETWORSTINDIVIDUAL(A);
10             if DIST(p, targetCell) < DIST(ind, targetCell) then
11                 A.insert(p) ;
12                 A.remove(ind) ;
13             else
14                 if DIST(p, targetCell) == DIST(ind, targetCell) then
15                     if p.behaviour < ind.behaviour then
16                         A.insert(p) ;
17                         A.remove(ind) ;
18                     else
19                         if p.behaviour == ind.behaviour & p.sparse > ind.sparse
                            then
20                             A.insert(p) ;
21                             A.remove(ind) ;
22                         end
23                     end
24                 end
25             end
26         end
27     end
28 end
29 return A ;
```

When the archive is not yet full (i.e., it contains less solutions than its predefined target size), DEEPATASH-LR adopts an inclusive approach towards candidate individuals. All individuals that reach the target cell or the neighboring feature map cells are included in the archive (lines 6-8). Otherwise, if the archive reaches its maximum capacity, the new candidate input competes with the worst individual currently present in the archive, i.e., the individual stored in the archive with the highest distance to the target and (for equal distances to the target) the lowest sparseness (line 9).

At the end of the search process, the FILTERMISBEHAVIOURS function (line 27 of Algorithm 4) is executed to filter and retain only the misbehaviour-inducing inputs in the archive. Since the archive may also contain correctly-behaving inputs, this filtering step is necessary to focus solely on the inputs that cause misbehaviours in the ADS under test.

6.4.5 Mutation

The MUTATE function (line 13 in Algorithm 4) mutates an existing input to generate a new input. This operator applies a perturbation, uniformly sampled in a customisable range, to the input model's control parameters. More specifically, DEEPATASH-LR mutates the road geometry by applying a displacement to the coordinates of the model's control points. Each time an input is mutated, DEEPATASH-LR verifies that the mutant complies with the domain-specific validity constraints. In particular, DEEPATASH-LR verifies that the mutant is different from its parent and it is a valid road, i.e., (1) the start point and the end point of the road should be different, (2) the road should fall within a square bounding box of fixed size, and (3) a road should not self-intersect. If any of these checks fails, the mutation operator is applied repeatedly, until a valid input is obtained.

6.4.6 Population Management

DEEPATASH-LR starts its search from an initial population of size *popsize*, which is obtained by selecting from a larger pool of inputs, named *seeds*. Function INITIALISE-POPULATION (line 2 in Algorithm 4) selects the *popsize* individuals closest to the target from the seeds S. More specifically, to generate the seed pool S, we randomly generate valid roads, i.e., roads with different start/end points that do not self-intersect and are entirely contained within a squared bounding box of fixed size.

6.5 Experimental Evaluation on ADSs

6.5.1 Research Questions

The goal of our evaluation is to understand the effectiveness of our approach in generating misbehaviour-inducing test inputs with the desired features for ADS. In particular, we compare two alternative surrogate model candidates, assess different alternative configurations of DeepAtash-LR, compare the best DeepAtash-LR configuration with an existing state-of-the-art test generator (DeepHyperion-CS), and investigate the usefulness of the generated test inputs for improving ADSs. Therefore, we answer the following research questions:

RQ0 (DNN vs LR): *Which type of surrogate model is more efficient and effective?*

A crucial aspect of our approach involves the careful selection of the surrogate model, which is tailored to the specific characteristics of the domain. To this aim, we explored the possibility of employing either Linear Regression (LR) or Deep Neural Networks (DNN) as surrogate models to predict the behaviour of the driving agent. LR fits a linear model with coefficients $(\beta_0, .., \beta_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip})\right)^2$$

where n is the number of observations, p is the number of features, y_i is the observed target for the ith observation, and x_{ij} is the jth feature value for the ith observation. We adopted three separate LR models for predicting two behavioural features values and the value of closeness to misbehaviour fitness.

As DNNs are general function approximators even for non linear functions, we considered two distinct DNN architectures. The former consists of 3 dense layers with *Sigmoid* activation, aimed to predict the closeness to misbehaviour. In fact, the *Sigmoid* is a nonlinear function that supports also negative values. The latter consists of 3 dense layers with *ReLU* activation, aimed to predict each behavioural feature. We use a *ReLU* activation as it is monotonic and positive and, thus, it works well for values between 0 and infinity. While there are other alternatives for designing the architecture of the surrogate models, we chose a simple 3-layered architecture as it is not a complicated task and because we wanted to keep our approach efficient, i.e., with low training and prediction time.

RQ1 (Surrogate Model): *Does the surrogate model improve the effectiveness of the original tool,* DeepAtash*?*

This RQ assesses whether the use of the surrogate model impacts the effectiveness of focused test generation in DeepAtash-LR. Specifically, our objective is to compare the performance of DeepAtash-LR with the previous version, DeepAtash, which does not utilize surrogate models.

RQ2 (Single vs Multi Objective): *Is* DeepAtash-LR *mode effective in the single or in the multi objective configuration for generating focused test inputs?*

DeepAtash-LR can be alternatively configured with single- or multi-objective search strategies, as explained in Section 6.1.4. This RQ aims at comparing the effectiveness of such two alternative configurations.

RQ3 (Comparison): *How does* DEEPATASH-LR *compare with the state-of-the-art tool* DEEPHYPERION*?*

In this RQ, we are interested in whether DEEPATASH-LR outperforms DEEPHYPERION-CS in generating test inputs in proximity of and within the target cell. We compare the best performing DEEPATASH-LR configuration (obtained from RQ2) against DEEPHYPERION-CS, as the latter is the only state-of-the-art test generator that targets the feature space at large by means of an illumination search algorithm. DEEPHYPERION-CS tries to cover all feature combinations and thus it is more likely to produce inputs on the selected target than e.g. random techniques, which may produce few or no inputs on the target, making the comparison with DEEPATASH-LR impossible. To the best of our knowledge, no other DL test generator is *focused*, i.e., capable of targeting a specific region of the feature space. Moreover, DEEPHYPERION-CS (Chapter 5) is a model-based test input generator that is applicable to BEAMNG and shares the same input representation and mutation genetic operators as DEEPATASH-LR. Such similarities allow for a fair and unbiased experimental comparison by eliminating confounding factors, which helps us to assess the specific and isolated contribution of our focused algorithm and the associated surrogate model (DEEPATASH-LR) to the test input generation process.

RQ4 (Usefulness): *Can the test inputs generated by* DEEPATASH-LR *be used to improve the ADS system under test?*

To investigate the usefulness of DEEPATASH-LR in a common DL usage scenario, we simulate a situation where a dev2op data shift is observed. This means that a particular feature combination is frequently encountered during the operation of the DL system, but it was inadequately represented or not present at all during the system's development phase, e.g., in the original training data. In this context, DEEPATASH-LR serves as a valuable tool for testers to address this data shift and target specific feature values of interest. By generating test inputs that focus on underrepresented and critical feature combinations, testers can effectively fine-tune the DL system. In this way, the generated test inputs are used to improve the quality of the DL system. Obviously, testers should assess also that fine-tuning does not introduce regressions or unintended side effects, e.g., the system may learn how to deal with the new test inputs, but might "forget" the correct behaviour for inputs belonging to the original training set.

6.5.2 Metrics

In this study, we define feature maps by considering high-level features that effectively characterize a self-driving scenario from the input and behavioural viewpoint. Specifically, we resorted to the features proposed in the Chapter 5. These features and the metrics to measure them were obtained by adopting a systematic methodology for feature definition and metric identification (Section 5.1). For this work, we chose the following features that cover both structural and behavioral attributes of the test inputs, thereby providing a comprehensive assessment of the driving agent's quality.

- *Standard Deviation of Steering Angle (StdSA)*: measures the level of activity exhibited by the driving agent on the steering wheel and can be used to quantify passenger comfort and driving quality;
- *Mean Lateral Position (MLP)*: represents the ego-car's positioning within the right lane. It is computed as the mean distance between the center of the car and the right lane margins;
- *Turn Count (TurnCnt)*: corresponds to the number of direction changes between consecutive road segments, with a requirement that the angle of change be at least 5°.;
- *Maximum Curvature (Curv)*: provides insight into the severity of the turns composing each road, by computing the reciprocal of the minimum road turn radius.

To compare alternative surrogate models, we use Mean Squared Error (MSE). This metrics are commonly used to evaluate the performance of ADS components in offline mode, i.e., without performing simulations. Given a set of predictions $\hat{y}_i$ and the corresponding expected values y_i for n data points, MSE is computed as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{6.6}$$

A lower MSE indicates that the model's predictions are closer to the expected values, suggesting a better fit of the model to the data. In addition, we report the training time of each model to assess their efficiency.

We evaluate DEEPATASH-LR's effectiveness by counting the inputs that fall within or close to the target cell. To this aim, we measure the *Tests Close to the target (TC)*, a metric that quantifies the number of generated misbehaviours in the proximity of the target feature map's cell, i.e. the solutions stored in the archive with distance to the target lower than or equal to 1. Additionally, we assess DEEPATASH-LR's ability to cover the target cell by computing the number of *Tests on Target (TT)*, i.e., the number of misbehaviours that fall exactly within the boundaries of the target cell. To evaluate the diversity of test inputs, we resort to the *Tests Close to the target Diversity (TCD)* and *Tests on Target Diversity (TTD)* metrics (see Section 6.2.2).

We evaluate DEEPATASH-LR's usefulness by assessing the driving model's performance after fine tuning it on DEEPATASH-LR's inputs. We consider both offline and online evaluation. Offline evaluation involves testing the DL model using pre-collected data without real-time interaction, while online evaluation involves deploying the DL model in a real-time, interactive environment, typically using a simulator.

MSE is particularly useful for assessing regression tasks, in which the goal is to predict continuous numerical values. We measure MSE before and after fine tuning the model in offline mode. To evaluate the model in online mode, we measure *Success Rate* (SR) which indicates the ability of the self-driving car to drive on the road without any failure

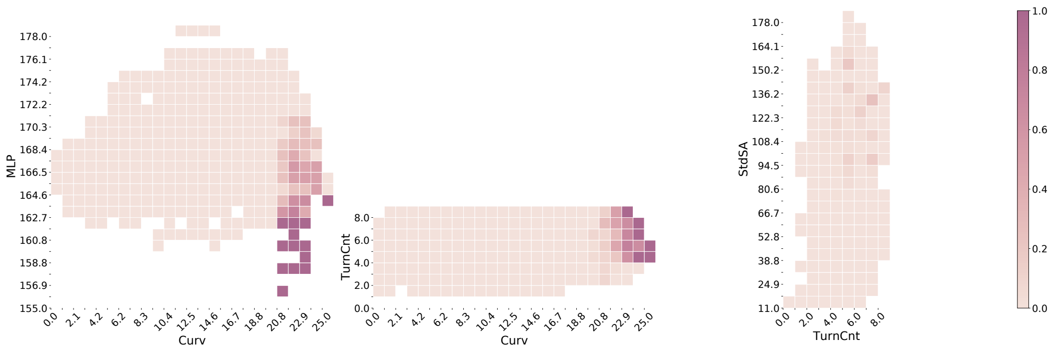


Figure 6.4. Average Misbehaviour Probability (AMP) maps generated by DeepHyperion for BeamNG. The axes quantify different features. The cells report the probability of exposing a misbehaviour for the corresponding feature value combinations, i.e., darker colors correspond to higher misbehaviour probabilities.

(i.e. without driving out of the boundaries of road). Online evaluation is conducted also to verify that the model successfully passes the regression test scenarios [53, 136].

6.5.3 Experimental Procedure

We evaluate DeepAtash-LR on a popular, safety-critical DL-based ADS. Specifically, we considered the DAVE-2 end-to-end driving agent, which utilizes the DL architecture developed by Bojarski et al. [17]. The objective of DAVE-2 is to address a regression task by predicting steering angles based on images captured by the on-board camera, ensuring the ego-car remains within the right road lane. DAVE-2 has been adopted as subject system by a large number of studies on ADS and DL system testing [70, 75, 138, 119, 61, 159, 161, 135].

To answer our research questions, we ran DeepAtash-LR in the three evaluation scenarios outlined in Section 6.2.3, along with DeepHyperion-CS, on the subject system. We focused our analysis on a subset of three pairs of features, chosen from the total of six possible pairs, due to practical considerations. Specifically, the cost of conducting complex and resource-intensive driving simulations escalates significantly. To ensure a controlled and effective evaluation process, we carefully selected three representative and diverse pairs of features that were likely to provide valuable insights into the performance and behaviour of the DL system. In particular, we chose a combination of two structural features (i.e., *Curv-TurnCnt*) and two combinations consisting of a structural and a behavioural feature (i.e., *TurnCnt-StdSA* and *Curv-MLP*). These selections are particularly interesting because for these combinations the exploration achieved by DeepHyperion-CS is not as extensive as for other feature combinations (which justifies a focused approach) and because they contain a lower number of misbehaviours.

For each evaluation scenario, the first step is the selection of the target cell from

Table 6.5. Hyperparameters used in the experiments

Parameter	BeamNG
seed pool size	80
initial pop size	48
population size	10
time budget (s)	36000
repopulation upper bound	2
target archive size	10
number of epochs for retraining	20
learning rate for retraining	0.0001

the misbehaviour probability maps generated by DeepHyperion-CS (see Figure 6.4). Specifically, we require that the chosen target cell contains a number of instances that falls below the average count observed across all cells within the feature map. This filtering mechanism ensures that cells with comparatively fewer occurrences are prioritized for selection, tackling the scarcity of data in such regions. Correspondingly, we randomly chose dark cells and grey cells with a coverage (i.e., number of individuals assigned to the cell) achieved by DeepHyperion-CS lower than the average cell coverage across all DeepHyperion-CS's runs.

Notably, this filter is not applicable to white cells, as they do not contain any existing data. For white targets, we selected cells that were not covered in the DeepHyperion-CS misbehaviour probability maps. However, since uncovered cells might correspond to unfeasible feature combinations, we randomly selected white cells that were adjacent (with a distance ≤ 1) to covered cells.

We ran DeepAtash-LR focusing on the identified target cells and collected the resulting archives of solutions. The hyperparameters of DeepAtash-LR were derived, whenever possible, from the experiments conducted with DeepHyperion-CS, reported in Chapter 5. We fine tuned these hyperparameters through a few preliminary runs to ensure that the target cells could be reached. The values of the hyperparameters are presented in Table 6.5.

We initiated the search process by randomly generating a pool of seeds, the quantity of which was defined as the *seed pool size*. These seeds serve as starting points for the search processes. From this pool of seeds, we selected the *initial pop size* inputs that were closest to the target cell. These chosen inputs constituted the initial population for the targeted test generation. By starting the search in proximity to the target, we focus the optimization process on relevant regions, thereby increasing the likelihood of generating inputs that trigger failures and possess feature values of interest.

To allow statistical analysis, we performed the same number of runs for each configuration of every test generator. We ran DeepAtash-LR 10 times for each type of target (i.e. dark, grey and white targets), for each feature combination (Curv-TurnCnt,

TurnCnt-StdSA, Curv-MLP), and for each considered search strategy (i.e. GA and NSGA-II). Since there were no dark target areas in the AMP of the $TurnCnt - StdSA$ feature combination, we excluded the dark target area specifically for this target type – feature combination pair. Hence, in total we performed $10 \times (3 \times 3 - 1) \times 2 = 160$ DEEPATASH-LR runs. For what concerns DEEPATASH, we started our experimentation considering one feature combination for each type of target. Correspondingly, DEEPATASH underwent execution with 2 search strategies for three pairs of target type and feature combination, resulting in a total of $2 \times 3 \times 10 = 60$ DEEPATASH runs. Since after collecting these results the superiority of DEEPATASH-LR was undoubtedly obvious, we preferred to save experimentation time and we did not run the remaining 5 pairs of target type – feature combination. DEEPHYPERION-CS is not a focused test generator, so it does not require to be executed for different target types. Consequently, we run DEEPHYPERION-CS 10 times for each of the three feature combinations ($10 \times 3 = 30$ DEEPHYPERION runs) In summary, the total number of runs performed was 250 (160 DEEPHYPERION + 60 DEEPATASH + 30 DEEPATASH-LR).

To ensure a fair comparison, we ran all tools with the same time budget (i.e., 10 hours). To assess the statistical significance of the comparisons between DEEPATASH and DEEPATASH-LR (RQ1), between different DEEPATASH-LR configurations (RQ2), and between DEEPATASH and DEEPHYPERION-CS (RQ3), we applied the Mann-Whitney U-test and measured the effect size by means of the Vargha-Delaney's $\hat{A}_{12}$ statistic [5].

In our experiments we used the best-performing surrogate model found by RQ0. Specifically, we ran DEEPATASH-LR on a predefined target cell with a time budget of 2 hours to collect the training data and trained the surrogate models, i.e., a LR and a DNN. Then, we assessed the surrogate models on a random pool of 40 valid roads. To allow statistical analysis, we repeated this experiment 10 times, using a different test set for each repetition.

To address RQ4, we performed fine tuning [12] of the DL model under test. The fine tuning process involved extending the model's training by conducting additional epochs under the same configuration (see Table 6.5). Such training was performed using the test inputs generated by DEEPATASH-LR that were close to (TC) and within (TT) the designated target regions. To this aim, we collected all the inputs generated by DEEPATASH-LR within and close to the targets. Then, we divided these inputs into two distinct sets, i.e., $training_{DA}$ and $test_{DA}$, 80% for training and the remaining 20% for testing. The combination of the original training set and $training_{DA}$ was used to fine tune the DL system. This choice reduces the risk of forgetting the learned task, by ensuring that both original training data and newly generated inputs are simultaneously available during training. We generated 10 random valid roads to serve as the test set. This test set and $test_{DA}$ were employed to evaluate the accuracy improvement of the fine-tuned DL system and verify if the driving agent exhibited a decline in handling inputs that were previously managed correctly before fine-tuning. To establish the statistical significance of the improvement in accuracy, we repeated the fine-tuning

Table 6.6. RQ0 - Mean Squared Error (MSE) and training time (Time) for DNN and Linear Regression (LR) as surrogate model for DeepAtash-LR. In each column, boldface indicates the minimum for MSE and time; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

	MLP		StdSA		Closeness to Misbehaviour	
Surrogate Model	MSE	Time (s)	MSE	Time (s)	MSE	Time (s)
DNN	2837.120	4.154	999.487	2.7436	**0.056**	2.461
LR	**4.823**	**0.059**	**793.185**	**0.000**	0.102	**0.001**

procedure 10 times for every run of DeepAtash-LR on each target cell. This allowed us to gather sufficient data and obtain reliable statistical results.

6.5.4 Results

RQ0: DNN vs LR

To choose the most suitable surrogate model for our purpose, we compared the candidate models, i.e., DNNs and LRs (see Section 6.5). Table 6.6 shows the results achieved by adopting DNN and LR models in terms of MSE along with the time needed to train them. LR achieved significantly lower MSE and training time than DNN for both behavioural features. Remarkably, LR achieved 588× lower MSE than DNN for the MLP feature. As fitness predictors, i.e., for the value of closeness to misbehaviour, DNN led to better predictions in terms of MSE, while training time were significantly worse than LR. Hence, we ultimately selected LR as the surrogate model for DeepAtash-LR.

The good performance of LR suggests that the relationship between the input features and the fitness values can be assumed to be approximately linear. LR is typically known for its computational efficiency and accuracy, when dealing with problems that exhibit linear trends in the data.

Summary: *Linear Regressors are more effective and efficient than DNNs as surrogate models for predicting behavioural features and the performance of the considered ADS.*

Table 6.7. RQ1 - Tests close to target (TC), tests on target (TT), tests close to target diversity (TCD), and tests on target diversity (TTD) by DeepAtash and DeepAtash-LR. In each row, boldface indicates the maximum metric values between DeepAtash and DeepAtash-LR, both with GA (resp. NSGA-II); underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

		DeepAtash				DeepAtash-LR			
		GA		NSGA-II		GA		NSGA-II	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-TurnCnt	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	**4.30 [0.50]**	**3.70 [0.40]**	**7.40 [0.90]**	**6.40 [0.80]**
Grey	TurnCnt-StdSA	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	2.40 **[0.70]**	**2.30 [0.70]**	**3.70 [0.70]**	1.70 [0.50]
White	Curv-MLP	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]	**4.80 [0.50]**	**2.90** [0.30]	4.70 **[0.50]**	**4.70 [0.50]**

Table 6.8. RQ1 - Number of iterations performed by DeepAtash and DeepAtash-LR in the same time budget. In each row, boldface indicates the maximum metric values between DeepAtash and DeepAtash-LR, both with GA (resp. NSGA-II); underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

		DeepAtash		DeepAtash-LR	
	Features	GA	NSGA-II	GA	NSGA-II
Dark	Curv-TurnCnt	116.10	132.70	<u>**187.30**</u>	<u>**183.10**</u>
Grey	TurnCnt-StdSA	104.70	122.70	<u>**155.20**</u>	<u>**196.80**</u>
White	Curv-MLP	117.60	114.40	<u>**312.80**</u>	<u>**284.80**</u>

RQ1: Surrogate Model

To assess the usefulness of the surrogate model, we conducted a comparative analysis between DeepAtash-LR and DeepAtash. Specifically, we considered only the three evaluation scenarios in which both test generators were focused on the same type of target for the same feature combination (see first two columns of Table 6.7 and Table 6.8).

Table 6.7 presents the results achieved by DeepAtash and DeepAtash-LR, using two distinct search strategies (GA and NSGA-II). As illustrated in the table, DeepAtash was not able to generate inputs close to the target, when operating without the support of a surrogate model. In fact, DeepAtash achieved zero test inputs both directly on the target (TT) and in close proximity to the target (TC) across all of its runs. Conversely, by integrating a Linear regression model within the search process, DeepAtash-LR exhibited a substantial enhancement. This improvement is manifested in the presence of diverse inputs generated, both in close proximity to the target (TC) and directly on the target (TT), across all feature combinations.

Furthermore, Table 6.8 reports the number of iterations performed by DeepAtash and DeepAtash-LR within the given time budget. This count of iterations serves as an indicator of the extent of the search process undertaken by each approach during the test input generation procedure. DeepAtash-LR executed a significantly higher number of search iterations compared to DeepAtash, for all targets (i.e., up to 199 more iterations for the white target). This increase can be attributed to the time-saving advantage gained by leveraging the surrogate model to avoid expensive simulations.

These results indeed affirm the crucial role of the surrogate model in directing DeepAtash-LR towards the target feature space through increased (surrogate) evaluations, thereby resulting in the creation of more focused test inputs, reaching the target.

Summary: *The integration of surrogate models into the focused test generation process is beneficial in scenarios where evaluations entail resource-intensive simulations. The adoption of surrogate models allowed* DEEPATASH-LR *to navigate the feature space with increased efficiency and efficacy, facilitating the production of diverse misbehaviour-inducing inputs in proximity of and within the target cells. In our case study, without surrogate model* DEEPATASH *did not reach any target cell and performed substantially less search iterations than* DEEPATASH-LR.

RQ2: Single vs Multi-objective

Table 6.9. RQ2 - Tests close to target (TC), tests on target (TT), tests close to target diversity (TCD), and tests on target diversity (TTD) by alternative DeepAtash-LR configurations. In each row, boldface indicates the maximum of each of the four metrics between GA and NSGA-II; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

		GA		NSGA-II	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-MLP	5.50 [0.50]	**5.50** [**0.44**]	**6.70** [**0.63**]	5.30 [0.34]
	Curv-TurnCnt	4.30 [0.36]	3.70 [0.31]	**7.40** [**0.65**]	**6.40** [**0.65**]
Grey	Curv-MLP	1.60 [0.19]	1.50 [0.12]	**5.80** [**0.71**]	**4.60** [**0.47**]
	Curv-TurnCnt	2.10 [0.31]	0.50 [0.10]	**4.10** [**0.51**]	**0.70** [0.10]
	TurnCnt-StdSA	2.40 [0.50]	**2.30** [**0.30**]	**3.70** [**0.54**]	1.70 [0.15]
White	Curv-MLP	**4.80** [0.36]	2.90 [0.26]	4.70 [0.36]	**4.70** [**0.44**]
	Curv-TurnCnt	4.10 [0.38]	3.30 [0.30]	**7.80** [**0.74**]	**7.20** [**0.63**]
	TurnCnt-StdSA	0.00 [0.00]	0.00 [0.00]	**2.00** [**0.50**]	0.00 [0.00]
	AVG	3.10 [0.32]	2.46 [0.22]	**5.27** [**0.58**]	**3.82** [**0.34**]

Table 6.9 reports the results achieved by the two configurations of DEEPATASH-LR, which adopt alternative search strategies. Specifically, we implemented GA as single-objective approach and NSGA-II as multi-objective approach. For each evaluation scenario detailed in Section 6.2.3, the table presents a row for every feature combination under consideration. It should be reminded that in the dark cell scenario, we computed the metrics only for two (Curv-MLP and Curv-TurnCnt) feature combinations, since there was not any dark cell in the TurnCnt-StdSA AMP.

For dark and grey targets, NSGA-II always produced the highest number of diverse inputs in close proximity to the target (i.e., TC and TCD values).

For white targets, NSGA-II achieved the highest TC, TT, TCD and TTD values for two feature combinations out of three. Remarkably, NSGA-II was the only DeepAtash-LR configuration able to generate inputs in proximity of the white target for the TurnCnt-StdSA feature combination.

These findings highlight overall the effectiveness of both search strategies (DeepAtash-LR with GA vs NSGA-II), with an advantage observed in favor of NSGA-II. The last row of Table 6.9 shows that on average, NSGA-II generated a higher number of diverse misbehaviours both close to and on the target cell.

The statistical significance of this performance difference was observed in all metrics, except for the test on the target diversity (TTD) metric, for which both strategies achieved statistically comparable results. By employing the NSGA-II algorithm, our tool optimized multiple fitness functions simultaneously. Such multi-objective approach allowed for a more comprehensive exploration of the feature space. Consequently, this led to an increased number of useful test inputs and provided a diverse set of driving scenarios.

Summary: *The multi-objective configuration of* DeepAtash-LR *generated a larger number of inputs in close proximity to and exactly on the target cell compared to the single-objective configuration. Moreover, the multi-objective configuration exhibited higher diversity of the generated inputs.*

RQ3: Comparison

Table 6.10 reports the results achieved by DeepAtash-LR and the existing approach DeepHyperion-CS. In this comparison, we focused on the DeepAtash-LR configuration featuring the NSGA-II multi-objective search strategy, as it had demonstrated superior performance in the previous research question.

As indicated in the first two rows, for all dark targets, DeepAtash-LR demonstrated its superiority by generating significantly more and more diverse inputs in close proximity to and exactly on the target cell. Remarkably, for Curv-MLP and Curv-TurnCnt feature combinations, DeepAtash-LR generated an average of 58.5× more misbehaviours on target compared to DeepHyperion-CS.

For most of the grey targets, DeepAtash-LR and DeepHyperion-CS showed statistically comparable performance. Notably, DeepAtash-LR outperformed DeepHyperion-CS by generating a significantly higher number of tests close and on the target for TurnCnt-StdSA and tests on the target for the Curv-MLP feature combinations, along with achieving higher diversity. This performance gap was statistically significant for TT (46× higher) and TTD (7× higher).

As regards targets for which DeepHyperion-CS was unable to generate failure-inducing inputs (i.e., white targets), DeepAtash-LR produced an average of up to 7.20 such inputs. This highlights DeepAtash-LR's ability to successfully cover regions in the feature space that were completely unexplored by DeepHyperion-CS. Specifically, for the TurnCnt-StdSA feature combination, DeepHyperion-CS failed to generate any

Table 6.10. RQ3 - Results achieved by the compared tools. Tests close to target (TC) and their diversity (TCD); tests on target (TT) and their diversity (TTD). In each row, boldface indicates the maximum of each of the four metrics; underline indicates values significantly higher than the remaining ones (p-value < 0.05, non-negligible effect size).

		DeepAtash-LR		DeepHyperion	
	Features	TC [TCD]	TT [TTD]	TC [TCD]	TT [TTD]
Dark	Curv-MLP	**6.70** [**0.64**]	**5.30** [**0.68**]	0.90 [0.10]	0.10 [0.02]
	Curv-TurnCnt	**7.40** [**0.79**]	**6.40** [**0.78**]	1.50 [0.40]	0.10 [0.02]
Grey	Curv-MLP	**5.80** [**0.60**]	**4.60** [**0.70**]	2.30 [0.31]	0.10 [0.10]
	Curv-TurnCnt	**4.10** [**0.55**]	**0.70** [**0.10**]	1.20 [0.2]	0.10 [0.05]
	TurnCnt-StdSA	**3.70** [**0.70**]	**1.70** [**0.50**]	0.40 [0.25]	0.10 [0.10]
White	Curv-MLP	**4.70** [0.48]	**4.70** [**0.50**]	1.00 [**0.52**]	0.00 [0.00]
	Curv-TurnCnt	**7.80** [**0.97**]	**7.20** [**0.90**]	0.30 [0.12]	0.00 [0.00]
	TurnCnt-StdSA	**2.00** [**0.50**]	0.00 [0.00]	0.00 [0.00]	0.00 [0.00]
	AVG	**4.86** [**0.59**]	**3.62** [**0.47**]	0.95 [0.30]	0.06 [0.08]

misbehaviours even in close proximity to the target, while DeepAtash-LR succeeded in producing an average of two diverse misbehaviour-inducing inputs.

The comparisons carried out across various evaluation scenarios consistently show that DeepAtash-LR outperformed DeepHyperion-CS, with statistically significant superiority observed in 60% of these comparisons. In the final row of Table 6.10, it is evident that, on average, DeepAtash-LR achieved better results than DeepHyperion-CS. The statistical significance of this performance difference was observed across all the considered metrics. In particular, DeepAtash-LR generated an impressive 60 times more inputs for the selected targets, underlining its substantial advantage. Our results demonstrate the capability of DeepAtash-LR in generating test inputs in feature space areas where the test generator DeepHyperion-CS can generate few or no inputs.

Summary: DeepAtash-LR *outperforms the state-of-the-art tool* DeepHyperion-CS *in generating misbehaviour-inducing inputs with specific target feature value combinations.* DeepAtash-LR *can effectively explore crucial areas of the feature space of the DL system, which might be overlooked by* DeepHyperion-CS.

Table 6.11. RQ4 - Mean Squared Error (MSE) and Success Rate (SR) on the original test set and on the test set generated by DeepAtash-LR, before and after fine tuning the DL system with the training partition of generated inputs. In each row, boldface indicates the minimum for MSE and maximum for SR; underline indicates values significantly lower/higher than the remaining ones (p-value < 0.05, non-negligible effect size).

	Test Set		DA Test Set	
	MSE before	MSE after	MSE before	MSE after
Curv-MLP		**0.0003**	0.0150	**0.0095**
Curv-TurnCnt	0.0006	**0.0003**	0.0301	**0.0139**
TurnCnt-StdSA		**0.0003**	0.0222	**0.0159**
	SR before	SR after	SR before	SR after
Curv-MLP		0.99		**0.75**
Curv-TurnCnt	**1.00**	0.93	0.00	**0.57**
TurnCnt-StdSA		**1.00**		**0.58**

RQ4: Usefulness

Table 6.11 shows the improvements of the driving agent, which were achieved through fine tuning using $training_{DA}$, the training partition of the inputs generated by DeepAtash-LR. The improvement is assessed by using two different test sets: (1) the test set made of 10 random roads and (2) $test_{DA}$, the test set partition generated by DeepAtash-LR. The "before" columns display the performance of the DL system on these two test sets; the "after" columns display the performance of the fine-tuned DL system.

Since we selected a high-quality ADS, its initial MSE on the original test set was quite low. Consequently, it was able to perfectly handle the driving task on such roads, achieving a SR of 1. On the other hand, the ADS's initial MSE on $test_{DA}$ was higher and its success rate was obviously 0, as this set consisted of misbehaviour-inducing inputs.

Table 6.11 (top) reports the results of the offline evaluation, in terms of MSE. Across all feature combinations, the fine-tuned ADS exhibited an improvement in its prediction accuracy. In fact, MSE significantly diminished after the fine tuning process, for both the test set and $test_{DA}$. Quite surprisingly, we improved the MSE also on the original test set despite the system's initial high quality. Therefore, we not only witnessed the absence of any signs of regression during the offline validation, but also noticed a slight improvement in MSE on the original test set. This might be due to an increased generalization capability induced by the additional training on inputs with under-represented feature combinations. Table 6.11 (bottom) reports the results of the evaluation in the simulation loop, in terms of SR. After fine tuning, we notice minimal regression in terms of the system's ability to successfully complete the driving task. In fact, the SR on the test set remains notably high and is statistically comparable to the perfect score achieved before the fine tuning process. On the other hand, the fine-tuned ADS demonstrated a significant improvement in its ability to drive on the roads belonging to $test_{DA}$ (i.e., SR = 63% on average).

Summary: DEEPATASH-LR *is useful to improve the performance of the ADS through fine tuning, by targeting feature combinations under-represented or unseen during development. The inputs generated by our tool can serve as a diverse training set, facilitating the enhancement of the performance of the ADS without compromising its success rate.*

Threats to Validity

External Validity: The selection of the experimental subject may pose a potential threat to external validity. Nevertheless, we mitigated this concern by choosing an ADS widely employed in SE research and adopting a driving simulator with precise physics simulation capabilities. Furthermore, we verified that our chosen subject could successfully manage all the driving tasks within a randomly generated test set.

The choice of targets introduces another potential external validity threat, as the obtained results may not necessarily generalize to different target selections. To address and mitigate this threat, we adopted a strategy for selecting three distinct types of targets, each corresponding to different usage scenarios.

Finally, the comparison between LR and DNN may not be representative of all the existing surrogate models. LRs have small number of parameters and can be trained with small sample of training data. DNNs are more general, as they can approximate any non linear function, but they have more parameters to train and, correspondingly, require more training data and time. Therefore, we believe that these two models cover the two most interesting classes of surrogate models, although we acknowledge that alternatives do exist.

Conclusion Validity: The stochastic nature of DL-based ADSs and search-based approaches may affect the results. To address this concern, we adopted a rigorous experimental methodology by running each experiment multiple times and conducting standard statistical tests to assess the significance of the obtained results.

6.6 Conclusions

DEEPATASH is the first automated focused test generator for DL systems. Our experiments show that it outperforms the state of the art in generating diverse misbehaviour-inducing inputs on predefined targets. Results show also that fine tuning a DNN on underrepresented inputs produced by DEEPATASH not only increases its prediction accuracy on them, but also its generalization ability on the whole input distribution.

We also proposed DEEPATASH-LR, a novel focused test generator for ADSs. DEEPATASH-LR employs a surrogate model as a proxy for actual system's execution, thus sidestepping the need for resource-intensive evaluations, which would involve complete simulations of the driving tasks on the candidate test scenarios. Our empirical study shows that the evaluation using surrogate model implemented within DEEPATASH-LR significantly improves the effectiveness of DEEPATASH in generating misbehaviour-inducing inputs in the proximity of the target features. Moreover, we obtained empirical evidence showing that the focused inputs generated by DEEPATASH-LR can be useful for improving the ADS through fine tuning.

6.7 Reproducibility

The code implementing DEEPATASH and DEEPATASH-LR, the dataset, and all the scripts to replicate the experimental evaluation are available online [163].

Chapter 7

Automated Feature Extraction for Testing Deep Learning Systems

Recent approaches based on illumination search, such as DEEPHYPERION-CS (Chapter 5), evaluate DL system quality by explicitly searching for critical, misbehaviour-inducing inputs with different features. Their output consists of *feature maps*, N-dimensional grids that represent the performance of generated inputs in the space of the relevant features. Feature maps proved to be extremely useful in several testing tasks such as test selection [105], test adequacy assessment [39, 13], and misbehaviour explanation [162].

A crucial problem of illumination search algorithms is the definition of features, as these are usually problem- and domain-specific. We introduced a systematic methodology for defining features within a domain of interest. This methodology involves multiple human experts with domain knowledge, who identify features (i.e., map dimensions) and metrics for quantifying such features (see Section 5.1). Additionally, we require human effort also for designing models of the input to be perturbed by mutation operators. Indeed, involving human experts contributes to more meaningful and understandable feature dimensions. However, the careful engineering required for defining features, metrics and input models is not trivial and may impose several limitations on the applicability of this testing approach.

We propose a novel approach, DEEPTHEIA, that tackles the limitations of the state of the art by introducing automated feature extraction and input perturbation operators based on modern generative DL, which greatly reduce the costs of illumination search. Specifically, DEEPTHEIA leverages the knowledge about the target domain automatically learned by a DNN to extract the features that better capture the main characteristics of test inputs. This information is easily accessible from the weights of the internal layers of a general-purpose feature extractor or the network under test.

We evaluated our proposed technique on two different popular image classification problems with increasing complexity, i.e., classification of handwritten digits and clas-

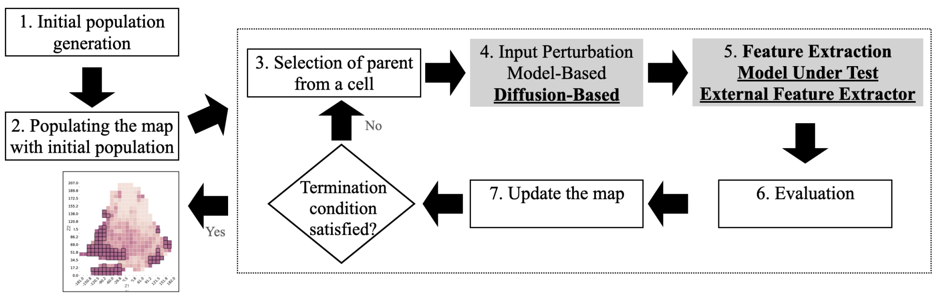

1. Initial population generation
2. Populating the map with initial population
3. Selection of parent from a cell
4. Input Perturbation
Model-Based
Diffusion-Based
5. Feature Extraction
Model Under Test
External Feature Extractor
6. Evaluation
7. Update the map
Termination condition satisfied?
No
Yes

Figure 7.1. Overview of DeepTheia

sification of real-world images. Our results show that, for both problems, DeepTheia produces feature maps that are highly discriminative in identifying the combinations of feature values that trigger misbehaviours of the DL system. In particular, DeepTheia outperforms the features manually defined by experts in problems where DeepHyperion-CS was applied with success. Remarkably, the features automatically extracted by DeepTheia perform well also for ImageNet [126], where human-defined features could not be defined. Furthermore, we conducted a study involving independent human assessors. Results show that the automatically extracted features produce cohesive groups of inputs mapped to the same cell, which indicates some degree of human interpretability of each feature map cell.

7.1 The DeepTheia Technique

As described in Chapter 5, the DeepHyperion-CS approach relies on human experts for feature definition and input model design, which requires domain expertise. In contrast, our new approach overcomes these limitations by automating feature definition and introducing input perturbation operators using generative DL. This results in a significant reduction in the costs associated with the illumination search process.

Figure 7.1 presents an overview of DeepTheia and highlights our proposed solutions for fully automating the test generation process. DeepTheia can perturb inputs by leveraging diffusion models, besides input models explicitly defined by test engineers (step 4). Unlike DeepHyperion-CS, DeepTheia extracts features automatically using its feature extractor component that exploits a transfer learning approach (step 5). In the next sections, we detail the key steps of DeepTheia.

7.1.1 Automated Feature Extraction

A crucial element of our approach is the automated feature extraction, as it directly influences the ability of the test generator to produce a diverse test suite. Indeed, the extracted features define the feature map, which is progressively populated with the

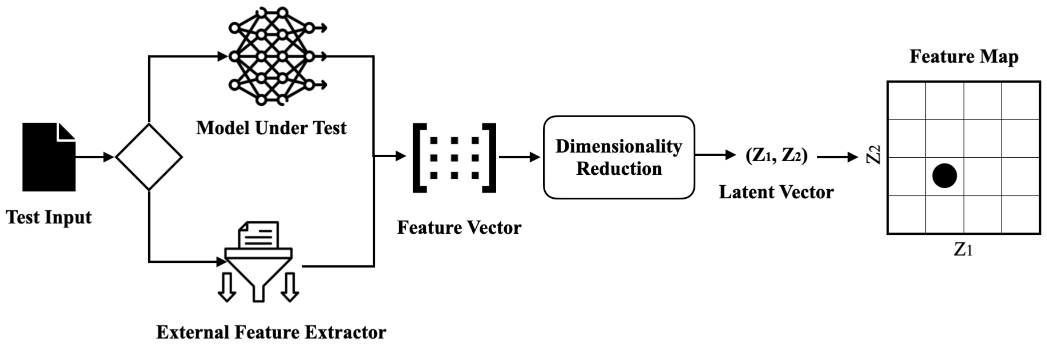


Figure 7.2. Automated feature extraction from test inputs

most promising inputs during exploration. The key characteristics of features required by illumination search are that they must be discriminative and quantifiable.

Figure 7.2 shows an overview of our proposed approach for extracting features to be used as dimensions of the feature map. The goal of feature extraction is to identify input characteristics that allow DeepTheia to cluster similar inputs together, hence providing a similarity based explanation for the causes of DL system's misbehaviour, when a cell contains misbehaviours. We automatically generate features using the following process: firstly, using a feature extractor we obtain feature vectors, i.e., vectors of numerical values that preserve the relevant information in the original input (Section 7.1.1). Then, we project features onto the latent space through dimensionality reduction, to obtain data points with reduced dimensions, while retaining the maximum possible amount of information (i.e., data variation; see Section 7.1.1).

Feature Vector Generation

As shown in Figure 7.2, there are two alternative ways to generate feature vectors: (1) the model under test itself; (2) an external feature extractor.

The *Model Under Test*, which is a DNN, already includes feature extraction layers that are necessary for its processing. These may be convolutional, pooling and fully connected layers, which can be used to extract the features associated with a given input [47, 142]. For instance, in image processing the main building blocks of the DNN are convolutional layers that extract visual features from the input. By adding pooling layers on top of convolutional layers, the model can identify such visual features. To use the model under test for feature extraction, we need to pre-process the data to reshape the input vector and generate a vector with the size required by the model. We then feed the pre-processed vector to the model and extract features. To obtain higher level features, the output of the last feature extraction layers is collected. For instance, such output can come from the last fully connected layer before the softmax layer of an image classifier. The result is an abstract feature vector of size N_c, i.e., the size of c-th layer of the DNN. In this way, we can extract high level features from an input without having to define and train any additional feature extraction model. On the other hand, we are bounded by the capabilities of the model under test. If this is not good at feature extraction, we will obtain poorly performing features.

Alternatively, we can apply a transfer learning based feature extraction method using an *External Feature Extractor* model. Transfer learning leverages knowledge from a general domain and applies it to a specific domain through the fine tuning of a pre-trained model. Beyond the considerable time savings associated with transfer learning, empirical evidence suggests that starting with a pre-trained model can yield superior performance compared to training from scratch, even when addressing a distinct problem [150].

There are several pre-trained models which are widely used for feature extraction in the literature [65, 4, 8, 102]. For instance, VGGNet is a convolutional neural network with multiple layers (i.e., 11 to 19 layers) for image recognition proposed by the Visual Geometry Group at the University of Oxford [131]. Its feature extraction component spans from the input layer to the final maximum pooling layer that outputs the feature matrix. In order to use an external feature extractor, the input vector must be pre-processed to fit the required input size and shape of the model. Then, the pre-processed input is fed to the model. In the case of VGGNet, the last pooling layer returns a feature matrix, which is flattened to generate the final feature vector.

Using any of the above approaches, the output of this step is a multi-dimensional feature vector that abstracts important characteristics of the input into a numerical embedding vector.

Dimensionality Reduction

In this step, we apply dimensionality reduction techniques to the high-dimensional feature vector generated in the previous step. The goal is to obtain a lower-dimensional vector in the *latent space*, a low-dimensional representation, inferred from the distribution of input data and preserving the similarity among inputs with similar features. In this work, each dimension of the latent space corresponds to one of the axes of DeepTheia's feature map.

The rationale behind reducing their dimensionality lies in the richness and diversity of feature vectors, which contain a plethora of information: attempting to find common characteristics among different inputs in such high dimensional space and generating discriminative feature maps is impractical without dimensionality reduction. In fact, in a high dimensional space, feature map cells tend to be sparse and scarcely populated, providing little additional information to the feature map user.

Hence, we use Principal Component Analysis (PCA) [37], a statistical method commonly used for dimensionality reduction and data visualization. It transforms high-dimensional data into a new coordinate system, where the axes are the principal components. These principal components are linear combinations of the original variables, and they are chosen to capture the space directions of maximum variance in the data. To this aim, we "train" a PCA model on each considered input dataset. PCA computes the N_p space directions (eigen-vectors) where the input domain is projected (with N_p the number of reduced dimensions, defined by the user). In this way, PCA summarizes the information content in a high dimensional dataset, by transforming a large number of attributes into a smaller one, while retaining the majority of the information (variance) present in the original attribute values.

Through PCA, we can generate a pre-defined number of feature dimensions (i.e. the number of components selected after PCA) while preserving the most important information of the inputs. Once PCA is trained, we can use it to determine the latent vector corresponding to each input feature vector.

Since the latent space is continuous, we obtain the feature map coordinate along each dimension through discretization, by scaling the latent feature value $ind.l_i$ using the scaling factors α_i:

$$x_i = \lfloor \alpha_i \cdot ind.l_i \rfloor \tag{7.1}$$

where $ind.l_i, \forall i \in [1 : N]$ indicates an individual's latent feature values. The granularity of the map, representing the number of cells in each dimension, is adjustable by changing α_i, which can be tailored for a given problem to achieve the desired level of discrimination in the feature map.

7.1.2 Input Perturbation

Illumination search algorithms use mutation as an evolutionary operator to introduce small, random changes to individuals (i.e., candidate solutions) in the population. It mimics the concept of genetic mutation in biological evolution. In the context of testing, mutation is used to generate new and potentially better solutions, by making incremental modifications to existing test inputs.

The purpose of mutation operators is to explore the solution space by generating different (and potentially better) test inputs. Specifically, DEEPTHEIA projects the solution space to a feature space with lower dimensionality, i.e., the feature map, and applies mutation operators to perturb inputs from the map cells, so as to explore the feature map at large.

Test input generators for DL [91, 154, 24, 26, 90, 119] typically apply small perturbations to initial seeds, i.e., inputs with known ground truth labels, under the assumption that such perturbations will not change the label. For instance, a simple method for input perturbation may apply small changes [112, 51, 143] directly to the input space, e.g., by modifying pixel values for images. While these techniques proved to be extremely effective in assessing the robustness of the DL systems against adversarial attacks, they are limited in testing the DNN performance in the presence of unexpected data, i.e., inputs not represented in the training set that may occur during system operation. To overcome such limitations, two main families of approaches gained popularity in DL testing, i.e., the manipulation of models of the inputs and generative DL. We applied these two input perturbation approaches to our case studies. For classification of handwritten digits from the MNIST dataset, we used model-based input perturbation presented in Section 5.2.1. In the following, we describe how we applied a diffusion-based input perturbation approach to our case study, classification of photo-realistic images from the ImageNet-1K dataset.

Diffusion-Based Input Perturbation

When the DNN input is an image, we can use generative DL to perform input perturbation. Generative DL models operate by reconstructing the underlying probability distribution of their training data as a low-dimensional latent space usually consisting of a normal multivariate probability distribution of parameters. This knowledge is then used to generate new inputs or to modify existing inputs that belong to the considered input domain.

Differently from model-based input perturbation approaches, generative DL models do not need human effort in designing manually the input model, as the probability distribution of data is automatically extracted from the inputs used for training. For this reason, generative DL models are particularly useful when an input model is not available and is difficult to craft, e.g., for feature-rich input datasets such as real images.

Recent work from the literature adopt Variational AutoEncoders (VAEs) [72] and Generative Adversarial Networks (GANs) [45]. However, these DL models may generate invalid inputs due to the lack of continuity in the latent space. GANs are known for their potentially unstable training and lack of diversity of generated inputs due to their adversarial training.

Overall, the quality of inputs generated by these techniques heavily relies on the quality of the training set and of the adopted generative model [120]. Hence, we prefer to adopt more recent diffusion models for generating image variations, which result in more realistic and valid images [25], by using domain-specific text prompts (i.e, a text prompt including the expected class label) that guarantee the preservation of the ground truth classification label with high probability.

Diffusion Models: Diffusion models [133] are probabilistic models designed to learn a data distribution, denoted as $p(x)$, through a gradual denoising process applied to a variable sampled from a Gaussian distribution. In particular, the sampling process initiates with a noisy sample x_t and progressively generates less noisy samples x_{t-1}, x_{t-2}, ... until arriving to a final sample x_0. At each time step t, there is a corresponding noise level, and x_t can be understood as a mixture of a signal x_0 with some noise ϵ, where the signal-to-noise ratio is determined by the time step t.

Instead of operating directly on the image, latent diffusion models (LDMs) [122] operate by repeatedly reducing noise in a latent representation space generated by a VAE. Like other categories of generative models, LDMs have the potential to characterize conditional distributions. Text conditioned (text-to-image) LDMs [25, 106] process a text prompt fed into the noise predictor U-Net [123]. Text-to-image models start from a random noisy image, while image-to-image latent diffusion models accept as input an image along with a textual prompt. The model starts by encoding the input image to the latent space. During the sampling step, the noise predictor U-Net takes the latent noisy image and the textual prompt as inputs and predicts the noise in the latent space, considering the image features described within the prompt. Then, it generates a new latent vector by subtracting the noise from the input latent vector. After repeating this sampling step a predefined number of times, the VAE decodes the obtained latent vector to generate the new image. Generating realistic images while retaining the style and semantic content of the input image is challenging for image-to-image LDMs, since the latent distribution is biased in comparison to a standard Gaussian distribution. For this reason, Zhang et. al. introduced an inference pipeline called Real-world Image Variation by Alignment (RIVAL) [158] for diffusion models, which is able to generate high-quality image variations by performing adaptive cross-image attention and latent distribution alignment in the denoising steps. Using the RIVAL pipeline, it is possible to generate image variations while maintaining semantic and style consistency with the seed image (i.e. the reference image).

DeepTheia's diffusion-based approach: DeepTheia generates image variations using a modern image-to-image diffusion model. This involves supplying the model with a reference image and a domain-specific textual prompt. In this way, we address our twofold objective: to use a reference image for mutation and to ensure label preservation.

In particular, we adopt a pre-trained LDM called *Stable Diffusion* [122] and fine-tune it by using images from the training set of the target class. While the pre-trained model lays a robust foundation, tailoring it to a specific dataset and task significantly enhances its efficacy, ensuring alignment with user-defined objectives and preserving style, format and other qualitative aspects of the given input domain through fine tuning [38, 125]. By using the RIVAL inference pipeline, we feed the LDM with the reference image (i.e., the image to be mutated) and a predefined domain-specific textual prompt. As output, the LDM generates a new input, which is a variation of the reference image. One can control how much the output image adheres to the textual prompt by determining the *Guidance Scale* parameter. Hence, higher guidance scale means less creativity capacity for the LDM. The *Sampling Steps* parameter can also be used to determine the number of iterations LDM performs to denoise the image. With each step, some noise is progressively eliminated, leading to an improvement in the output image quality. However, the greater the number of sampling steps, the longer it takes to produce an image. Using textual prompts beside the image as inputs for the diffusion model guarantees label preservation, as the prompt ensures the presence of the subject class in the generated image.

Figure 7.3 shows two example images generated by our diffusion-based mutator. Figure 7.3 (a) is a sample "pizza" image and (b) is the mutant image generated by LDM

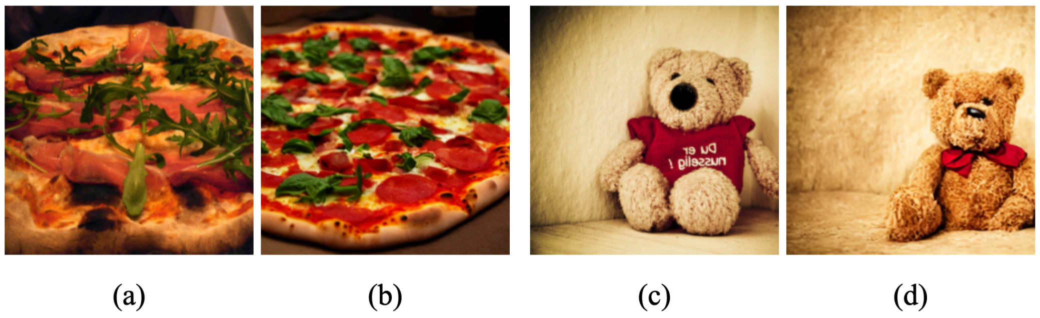


Figure 7.3. Diffusion-based input mutation. (a) original input of class "*pizza*"; (b) mutated input; (c) original input of class "*teddy*"; (d) mutated input.

starting from the previous sample image and the text prompt: "*A photo of pizza, best quality, extremely detailed*". Figure 7.3 (c) is a sample image labeled as "teddy", while (d) is the mutated image generated by LDM when feeding it with the sample image and the text prompt: "*A photo of teddy, best quality, extremely detailed*".

7.2 Experimental Evaluation

7.2.1 Research Questions

The goal of our evaluation is to understand whether the features automatically extracted by DeepTheia are effective for testing DL systems through illumination search. Therefore, we seek answers for the following research questions:
RQ1 (Feature Discrimination): *How are the features automatically extracted by* DeepTheia *able to discriminate failure-inducing inputs?*

Effective features should be able to define feature maps that identify the combinations of feature values that are likely to trigger a misbehaviour of the DL system under test. This insight could offer developers a deeper understanding of the root causes of misbehaviours. In fact, the presence of regions in the feature map (i.e., one or more adjacent cells) characterized by significantly high probabilities of misbehaviours can suggest that the input data clustered into these cells are prone to causing misbehaviours. Moreover, generation or acquisition of new data that fall into these cells can be useful to obtain more evidence about the observed failures and to possibly fix them (e.g., by re-training).

Metrics: We aim to assess whether the generated feature map M, defined by the automatically extracted features, is discriminative. Moreover, we aim to verify that the combination of DeepTheia's features with illumination search is effective as it enables a thorough exploration of the feature map. For the latter aspect, we measure the map coverage as number of *Filled Cells* (FC) in the map; for the former the *Average Cell Impurity* (ACI) of the map with respect to the behaviour of the inputs in each filled cell:

$$ACI(M) = \frac{\sum_{i=1}^{FC} 1-(p^2_{misb_i} + p^2_{correct_i})}{FC} \tag{7.2}$$

where p_{misb_i} and $p_{correct_i}$ are the probabilities of misbehaviour and correct behaviour of the model in the i_{th} filled cell of the map M, respectively. Based on Equation 7.2, the ACI value is between 0 and 0.5. A low value of ACI means that DEEPTHEIA can effectively discriminate the system's behaviour, by grouping in the same cells inputs with the same behaviour.

RQ2 (Understandability): *How understandable are the features automatically extracted by DeepTheia?*

Although features are automatically extracted (hence, not necessarily human interpretable), it would be useful if they were also able to group inputs in a way that is understandable to humans.

Metrics: The comprehensive evaluation of the understandability of our approach requires humans in the loop. Therefore, we performed a human study involving independent assessors to determine whether the group of images from a feature map cell are more cohesive (i.e., contains more similar images) than a group of randomly selected images. We report the cohesiveness rate for the groups of images from the same cell vs randomly selected images from different cells of the feature map.

7.2.2 Experimental Procedure

We evaluate our approach using two popular image datasets, i.e., MNIST and IMAGENET. These datasets are commonly employed in the literature to assess testing techniques for DL systems [87, 66, 117, 27, 8, 152] and enable two distinct image classification tasks. In particular, IMAGENET, with its 1k classes and large-size real images poses a challenging task to test generators. Due to the complexity of IMAGENET images, it is extremely difficult to define discriminative and understandable features. For each of these two subjects, we consider widely adopted, pre-trained DL models.

We trained a weaker model for MNIST to show how DEEPTHEIA performs as model quality varies. To obtain a weaker model we injected the "sub-optimal learning rate" fault from the taxonomy of real faults for DL [60]. Specifically, we maintained the same configuration but used a lower learning rate of 1×10^{-6}, resulting in a test accuracy of 88.12%.

To answer our research questions, we ran DEEPTHEIA against the considered subjects. As a baseline, we considered DEEPHYPERION-CS, the approach presented in Chapter 5. Since it can perform only model-based input perturbations and an input model is available only for the MNIST subject, we can compare our new tool with DEEPHYPERION-CS only on MNIST.

We consider the results achieved by DEEPTHEIA on the two considered subjects with two different feature extraction approaches, i.e., by using (1) the same model under

Table 7.1. Hyperparameters used in the experiments

Parameter	MNIST	IMAGENET
class/classes	5	Pizza, Teddy
initial pop size	800	100
time budget (s)	3600	10800
input perturbation type	model-based	diffusion-based
guidance scale	-	5
sampling steps	-	50
image size	28×28	224×224
model	ConvNet	ResNet50
framework	Tensorflow	PyTorch

test or (2) an external feature extractor. For each feature extractor, we trained its PCA extractor by using the inputs with the label of interest from the original training set and setting the number of components to be selected (which corresponds to the number of feature dimensions) to 2. We performed only one training of each PCA component for each considered subject as the output of PCA is deterministic.

Additionally, we report the results without test generation, exclusively considering inputs from the MNIST test set and the IMAGENET training set. Due to the insufficient number of inputs (50 inputs) for each class in the IMAGENET test set for generating the feature maps, we used inputs from the training set instead.

For MNIST, we used DEEPHYPERION-CS with three different combinations of the following manually defined features (see Section 5.1.3): (1) *Luminosity (Lum)*, i.e. number of pixels whose value is above 127; (2) *Orientation (Or)*, i.e. vertical orientation of the digit, obtained by computing the angular coefficient of the linear regression of the non-black pixels; (3) *Moves* (Mov), i.e., sum of the Euclidean distances between pairs of consecutive sections of the digit. Instead, on IMAGENET we do not report any results for DEEPHYPERION-CS as it is not applicable to this complex dataset, which is not equipped with manually defined/quantified features and with a model of the input data, needed to perform input perturbation.

To ensure a fair comparison, all the feature maps were generated with the same number of cells for each feature, i.e. 25 cells, and dimensions, i.e., 2. The extreme values defining the range for each feature are the ones observed across the runs of the tools.

To account for non-determinism, we ran each tool 10 times on both MNIST and IMAGENET. This allowed us to analyse the statistical significance of the differences between tools. We used the Mann-Whitney U-test and measured the effect size by means of the Vargha-Delaney's $\hat{A}_{12}$ statistic [5].

Table 7.1 presents the values of the hyperparameters we used for each tool. We configured DEEPHYPERION-CS according to the configuration suggested in the original paper. We empirically obtained the configurations for DEEPTHEIA through some preliminary runs. For MNIST, DEEPTHEIA randomly selects an initial population made of 800 inputs from the official MNIST test set, all belonging to the same class (i.e. digit "5"). For IMAGENET, DEEPTHEIA randomly selects 100 inputs from the IMAGENET official training set. We considered two different IMAGENET classes in our experiments (i.e. "pizza" and "teddy").

We used the same model-based input perturbation approach to manipulate MNIST inputs both with DEEPTHEIA and DEEPHYPERION-CS. In this way, we can effectively rule out all confounding factors and clearly compare the features automatically extracted by DEEPTHEIA with the ones defined by experts for DEEPHYPERION-CS.

For IMAGENET, we used our novel diffusion-based input perturbation approach, presented in Section 7.1.2. In particular, we used the pre-trained Stable-Diffusion V1.5[1] model provided by Runway[2] and fine-tuned it on an NVIDIA GeForce RTX 2080 Ti GPU machine using Dreambooth [125] for 3000 training steps. For the fine tuning, we used all the inputs in the IMAGENET training set belonging to the considered target class (i.e., pizza or teddy) and 200 images generated by the diffusion model itself with a domain-specific prompt (e.g. "A photo of pizza") designed to ensure label preservation. For the inference, we used the RIVAL pipeline[3] (the most recent approach for generating real and high quality images at the time of writing) with guidance scale 5 and sampling steps 50.

For the human study, we published two surveys (one for each subject) using the Mechanical Turk platform provided by Amazon[4]. Each survey consists of 11 questions to be answered by human assessors: 10 assessment questions (ASQ) and 1 attention check question (ACQ). Specifically, we randomly selected 10 cells from a feature map generated by DEEPTHEIA with the best performing feature extractor and generated plots with groups of 4 images from each cell. Then, we generated 10 groups of 4 random images from different cells (with a minimum mutual distance of 9, which was the maximum possible distance to have at least 10 different groups of random images) from the same feature map. In each ASQ, we showed the human assessor a group of images from one feature map cell and a group of random images and asked them *"Which group of images are more cohesive (i.e. contains more similar images)?"* The assessors were provided with three possible choices: they could indicate that either the first group of images (>) or the second group of images (<) is more cohesive, or the two groups have the same level of cohesiveness (=). Assessors were also provided some examples of cohesive vs random groups of images, to explain them how to carry out the task. To avoid bias, such examples come from an independent dataset (Fashion-MNIST). Figure 7.4 and Figure 7.5 show two sample questions from the human study for MNIST

[1]https://huggingface.co/runwayml/stable-diffusion-v1-5
[2]https://runwayml.com
[3]https://github.com/dvlab-research/RIVAL
[4]https://www.mturk.com

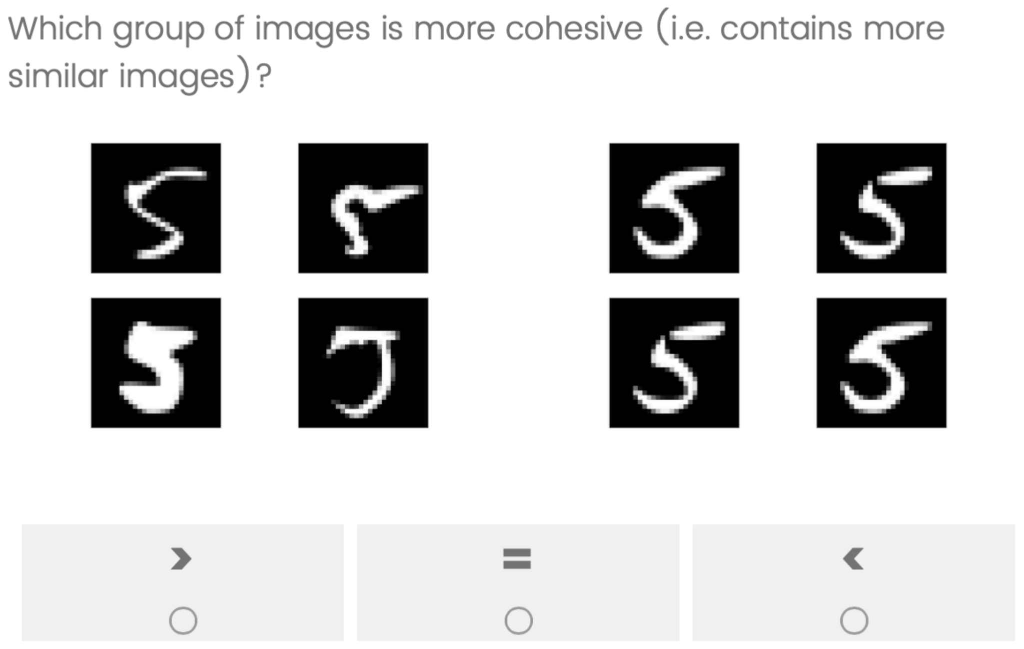

Which group of images is more cohesive (i.e. contains more similar images)?
>
=
<

Figure 7.4. sample human study question for MNIST. The group of images on the right are from the same cell of a feature map. The group of images on the left are selected randomly from different cells of the same feature map.

and ImageNet, respectively. For ACQ instead, we showed the human assessors the same groups of images, hence the two groups should be rated as equal in cohesiveness level (=). To ensure the quality of the answers we restricted the participation to the workers with approval rate above 95% and we only accepted answers from the users who passed the ACQ. We collected 80 answers from the human assessors, 40 for each case study.

7.2.3 Results

RQ1: Feature Discrimination

In this RQ, we investigate the discriminative capability of the features automatically extracted by DeepTheia. Table 7.2 and Table 7.3 report the results achieved by the considered tools for MNIST and ImageNet, respectively. Metric values are computed on the feature maps filled by either the original test/training sets or the inputs generated by the test generation approaches. This allows us to analyze the compared feature extractors both with and without integration with the test generators.

Columns 2 and 3 of Table 7.2 report the results obtained on the original MNIST test set, while columns 5 and 6 report the results by multiple runs of DeepHyperion-CS and DeepTheia. The results with and without test generation are in agreement. In

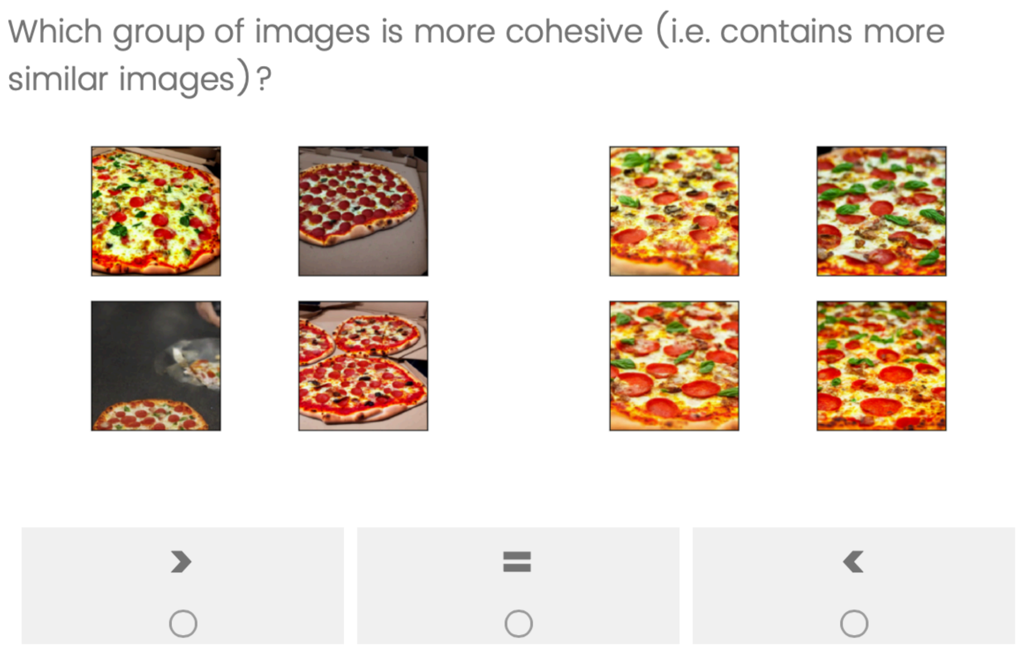
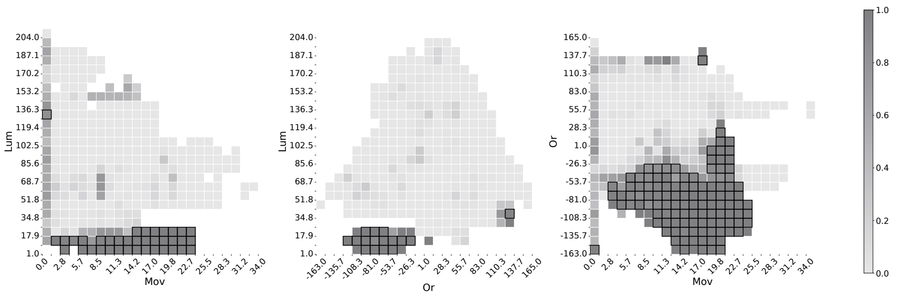

Which group of images is more cohesive (i.e. contains more similar images)?

Figure 7.5. sample human study question for ImageNet. The group of images on the right are from the same cell of a feature map. The group of images on the left are selected randomly from different cells of the same feature map.

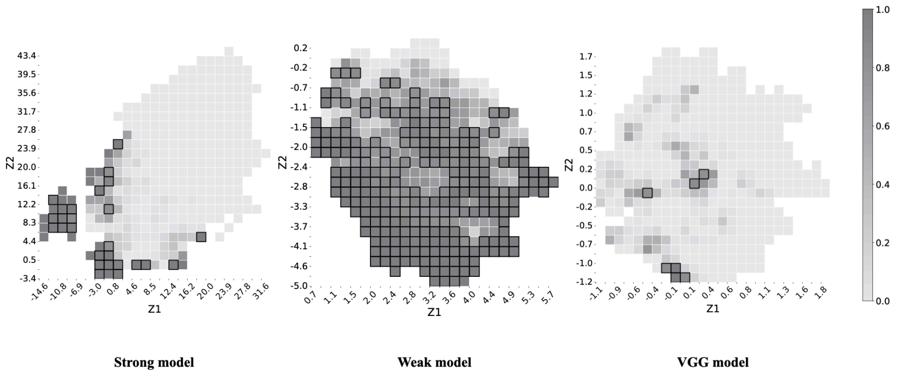
particular, the external feature extractor and the weak DNN generate maps that are always more significantly covered than those obtained by the human-defined features with large effect size. Moreover, the strong DNN model always achieves a significantly lower impurity (with large effect size and p-value < 0.05). This means DEEPTHEIA is better at grouping inputs with the same behaviours when a strong model is used as feature extractor. The $VGG16$ feature extractor achieves a significantly better ACI than the $Mov - Lum$ and $Or - Move$ feature combinations with large effect size, and has a comparable ACI with the $Or - Lum$ feature combination (p-value > 0.05), while covering the map more extensively than all of them (FC $= 346.3$).

We further analysed the results obtained by DEEPHYPERION-CS and DEEPTHEIA by comparing their Average Misbehaviour Probability (AMP) maps (see Figure 7.6). As shown in Figure 7.6 (b), maps obtained by the strong and external feature extractors have specific regions where the probability of misbehaviour is high (bold-bordered dark cells). This result is comparable with the AMP maps generated by DEEPHYPERION-CS (see Figure 7.6 (a)). Instead, the weak model failed to generate discriminative feature maps as it produces multiple regions with high misbehaviour probability scattered across the feature space.

As for IMAGENET, Table 7.3 shows that both automated feature extractors generated



(a) DeepHyperion-CS



(b) DeepTheia

Figure 7.6. Average Misbehaviour Probability (AMP) maps generated by (a) DeepHyperion-CS and (b) DeepTheia for MNIST. The axes quantify different features. The cells report the probability of exposing a misbehaviour for the corresponding feature value combinations, i.e., darker colors correspond to higher misbehaviour probabilities.

Table 7.2. RQ1 - Number of Filled Cells (FC) and Average Cell Impurity (ACI) of DeepTheia and DeepHyperion-CS for MNIST using different Feature Extraction (FE) models; best results in boldface.

	Test set		DeepHyperion-CS	
Features	FC	ACI	FC	ACI
Mov-Lum	93	0.006	269.2 ± 7.1	0.070 ± 0.007
Or-Lum	217	0.007	288.6 ± 7.6	0.031 ± 0.005
Or-Mov	88	0.005	279.2 ± 10.7	0.083 ± 0.008
	Test set		**DeepTheia**	
FE	FC	ACI	FC	ACI
Strong DNN	225	**0.004**	262.5 ± 8.2	**0.028** ± 0.004
Weak DNN	**278**	0.196	**357.1** ± 5.1	0.169 ± 0.010
VGG16	272	0.008	346.3 ± 9.3	0.052 ± 0.007

Table 7.3. RQ1 - The number of Filled Cells (FC) and Average Cell Impurity (ACI) of DeepTheia for ImageNet using different Feature Extractors (FE), for two different classes, i.e., "Pizza" and "Teddy"; best results in boldface

		Training set		DeepTheia	
Class	FE	FC	ACI	FC	ACI
Pizza	ResNet50	**276**	0.048	146.4 ± 5.1	0.021 ± 0.004
	VGG16	272	**0.039**	**185.4** ± 6.1	**0.011** ± 0.004
Teddy	ResNet50	229	0.100	**163.9** ± 7.5	0.061 ± 0.015
	VGG16	**251**	**0.052**	159.2 ± 6.1	**0.047** ± 0.009

discriminative maps, with low ACI both on training set and on tests generated by DeepTheia. In particular, the features extracted by the VGG16 model showed better ACI values with statistical significance for both subjects, while achieving a map coverage higher than (p-value < 0.05 for the class *Pizza*) or comparable to (p-value > 0.05 for the class *Teddy*) the features extracted by the model under test (Table 7.3 columns 5 and 6).

Singletons in feature map cells artificially decrease the value of ACI, because their impurity is by definition 0. To make sure that our results are not influenced by some unbalance in the occurrence of singletons, we analysed their prevalence and found it consistently around 40% with both $ResNet50$ and $VGG16$ feature extractors.

Summary: *Automatically extracted features result in highly discriminative feature maps (ACI < 0.04), enabling* DEEPTHEIA *to cover the feature space extensively (FC > 200). External feature extractors (VGG16) achieved superior or comparable performance as internal ones.*

A major implication of this study for practitioners is that not only automated feature extraction is possible and results in discriminative maps, but also that general purpose feature extractors, independent of the model under test, can be used for feature map construction. This relieves developers from the need of a strong model as feature extractor, which might not be available in the initial development phase, when the model might be still weak.

RQ2: Understandability

Table 7.4. RQ2 - Human assessment of Feature map vs Random based on cohesiveness, highlighted with best results.

Subject	Feature map	Random	No difference
MNIST	**78.25%**	4.00%	18.50%
IMAGENET	**78.25%**	8.75%	13.00%

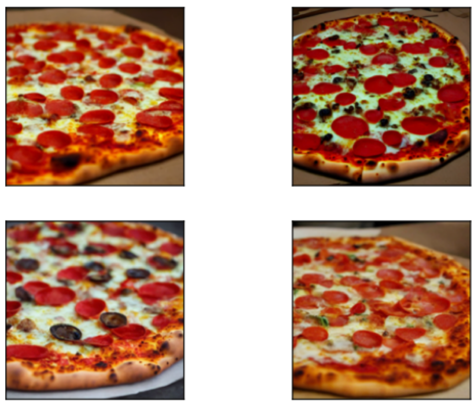
Table 7.4 reports the results of our human study on the cohesiveness of DEEPTHEIA's feature maps. In the *Feature map* column, we report the average percentage of the crowdworkers who identified the group of images from the same cell as more cohesive. The *Random column*, reports the percentage of answers where the randomly selected group of images was considered more cohesive. The last column indicates the average percentage of crowdworkers who considered a similar cohesiveness level between the two groups.

Overall, crowdworkers were able to perceive the higher cohesiveness of feature map cells (more than 78%). Despite the variety of IMAGENET images, the cohesiveness of the feature map cells was clear for the large majority of the assessors. Statistical significance of the cohesiveness difference between one feature map cell and multiple random cells was assessed by applying the Mann-Whitney U-test: a significantly higher percentage (78.25%) of assessors chose the one cell images as more cohesive than the random cell ones, with p-value < 0.05 and large effect size.

Figure 7.7 shows image groups used in the human study: the images selected from one feature map cell are clearly similar among them (see Figure 7.7 (a) and (c)), while random images from different cells are more diverse (see Figure 7.7 (b) and (d)).

(a) Selected from one cell of the feature map of MNIST

(b) Randomly selected from different cells of MNIST


(c) Selected from one cell of the feature map of ImageNet

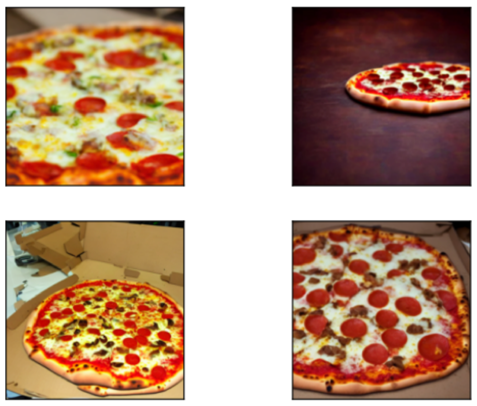

(d) Randomly selected from different cells of ImageNet

Figure 7.7. Sample groups of images used for the human study.

Summary: *The features automatically extracted by* DeepTheia *are associated with a perception of high cohesiveness in human assessors. The automatically generated feature map cells contain cohesive groups of images.*

A major implication of this study for practitioners is that the presence of a high proportion of misbehaviour-inducing inputs in a given feature map cell is to some extent human-interpretable. In fact our study shows that misclassified images assigned to the same cell form a cohesive group of images that share substantial similarity. This may possibly point to some human-understandable reason for the misbehavior, which might trigger proper corrective actions (re-training on real-world images with such features).

Threats to Validity

External Validity: A potential threat to external validity is the selection of the experimental subjects and datasets. To mitigate this threat, we chose two diverse image datasets with increasing complexity that have been widely adopted in the literature.
Conclusion Validity: The inherent stochasticity in DL and search-based approaches introduces variability in the results. To mitigate this, we employed a rigorous experimental methodology, running each experiment multiple times. We further applied standard statistical tests to evaluate the significance of the observed differences.

7.3 Conclusion

We introduced DeepTheia, a novel test generator for DL systems based on illumination search. It automates the extraction of relevant input features using pre-trained models, overcoming limitations of existing illumination-based tools by eliminating the need for human experts to define features.

DeepTheia shows significant improvements in the discriminative power of feature maps, while preserving their cohesiveness and understandability, with respect to expert-aided illumination search. Additionally, our novel mutation operator based on diffusion models enables the generation of valid tests for complex image classification tasks, while ensuring label preservation.

7.4 Reproducibility

The code implementing DeepTheia, the dataset, and all the scripts to replicate the experimental evaluation are available online [165].

Chapter 8

Comparison with Explainable AI Approaches

The lack of explainability of DL models may impact the trust in their predictions and consequently hinder the adoption of such systems. A common issue of DL systems is that they may take wrong decisions based on spurious correlations or biases in the training data, e.g., classifying wolves images as huskies due to the presence of snow on the background rather than leveraging the characteristics of the animal [115].

Multiple techniques address the aforementioned issues and explain DL systems' (mis-)behaviours. Existing explanatory techniques characterise the misbehaviour-inducing inputs either at a low level, by identifying the raw input elements (e.g., image pixels) that are most relevant for the prediction, or at a high level, in terms of abstract features that characterise the input as a whole. We will refer to these techniques as follows:

Low Level Explanations identify a portion of the input as relevant to a specific DL prediction. Such identification is performed directly in the input space of the DL model. Low-level techniques can provide explanations with different aggregations, reporting, e.g., atomic input elements (e.g., pixels/words for an image/text) or contiguous regions of input elements (e.g., sets of contiguous pixels/words). We considered two techniques with different aggregations, representing the state of the art in the eXplainable AI (XAI) community: LIME [115] and INTEGRATED GRADIENTS (IG) [139].

High Level Explanations identify relevant features that characterise the whole input. Such identification is performed in the feature space, a manually-defined abstraction of the input space. As high-level technique, we considered FEATURE MAPS (FM) that characterise inputs through human-interpretable features defined by domain experts (see Chapter 5).

We conduct an empirical study to investigate the similarity between explanations provided by low- and high-level techniques for the same sets of inputs. In this way, we can understand whether and to what extent different techniques are overlapping or complementary in explaining DL misbehaviours. Explanatory techniques provide an

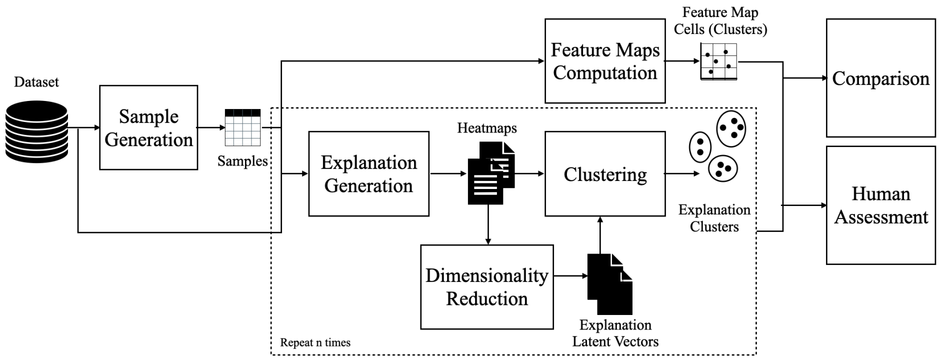


Figure 8.1. An overview of our evaluation pipeline

input characterisation that practitioners (e.g., software testers) can use to understand why an input triggered a misbehaviour [99]. For this reason, we involved human experts in our study to assess the understandability of the produced explanations, i.e., whether human assessors consider the explanations as effectively pointing to a plausible cause of the misbehaviour.

By leveraging input clustering and a novel ad-hoc similarity metric, our results show that high- and low-level techniques provide different explanations, i.e., they partition the same inputs in different ways. For most of the inputs, human experts chose either a low- or a high-level technique as effective in explaining misbehaviours. These results show that high- and low-level explanations are highly complementary. Results also show that both explanations often do not match human judgement, especially when explaining misbehaviours for image classifiers, hence demanding for novel XAI techniques that can cover the mis-explained cases. The lack of explainability of DL models may impact the trust in their predictions and consequently hinder the adoption of such systems.

8.1 Comparing High- and Low-level Explanations

To compare high-level and low-level explanations, we designed the evaluation pipeline shown in Figure 8.1. Initially, we gather the inputs to be explained. Such inputs can be either obtained directly from an existing dataset or can be automatically generated by test generators (*Sample Generation* step). For each input, we obtain high-level explanations in the *Feature Map Computation* step, i.e., we compute the FM cells it belongs to. In this way, each FM cell represents a cluster of inputs sharing similar high-level features. To obtain low-level explanations, we perform the *Explanation Generation* step, in which we apply low-level techniques that generate heatmaps. Moreover, we perform a *Dimensionality Reduction* step on low-level explanations to generate lower-dimensional latent vectors. A *latent vector* is the projection of an input onto the latent space. Using latent vectors allows us to avoid high variability and sparseness in the high-dimensional heatmap explanations. Indeed, a sparse, highly variable representation of the inputs would prevent the construction of meaningful clusters of the inputs, as in the higher-dimensional space all distances/similarities tend to degenerate to the extreme values, giving raise to degenerate clusters. Since both low-level explanatory techniques and dimensionality reduction approaches are non-deterministic, we repeat the low-level explanation generation multiple times. To make the explanations provided by low-level techniques comparable to the high-level ones, we cluster low-level explanations by using a *Clustering* technique. In the *Clustering* step, we group together similar heatmap explanations, or the corresponding latent vectors (to address the sparseness problem). In the *Comparison* step, we assess how similar the clusters generated by high-level and low-level techniques are, based on our custom definition of a *Gini similarity* metric, which is close to zero when the elements grouped together by one technique are scattered across many clusters produced by the other technique and is equal to one when the elements grouped together by one technique are all in the same cluster produced by the other technique. Finally, we submit a questionnaire to experts in order to evaluate the understandability of explanations in the *Human Assessment* step. In the following, we provide a detailed explanation of each step of our evaluation pipeline.

Sample Generation: Since we are interested in investigating DL misbehaviours, we resort to failure-inducing inputs artificially crafted by test generators, besides the ones from the original dataset. In this way, we can collect enough misbehaviour-inducing inputs even for robust DL models for which there are only a few misbehaviours in the original test set. By considering both types of inputs, we perform our study on a sufficient number of misbehaviours, covering a diversified variety of samples. We do not use the train set because we mimic the testing phase, when developers evaluate a DL model against new, unseen inputs

FM Computation: To provide high-level explanations, we used FM. FEATURE MAPS can be generated with different feature combinations and numbers of dimensions. For the sake of completeness, we considered all possible numbers of dimensions and feature combinations when computing the FM. In this way, we can discuss the similarity between high- and low-level explanations at increasing FM dimensionality (e.g., when moving from one isolated feature to a combination of multiple features). For each input obtained in the Sample Generation step, we compute its corresponding feature values, so we can assign each input to the corresponding FM cell, i.e., the map cell whose value intervals contain the measured input feature values. We use the non-empty FM cells, i.e., those containing at least one misbehaviour-inducing input, as high-level explanation clusters.

Explanation Generation: In this step, we consider two popular XAI techniques as our low-level explanatory techniques: IG for fine-grained explanations and LIME for coarse-grained explanations. We apply the two considered XAI techniques separately on the generated inputs and extract heatmaps as vectors representing the relevance of input elements/regions.

Dimensionality Reduction: Beside the heatmap explanations in the original input space, we consider explanations projected onto a latent space, i.e., latent vectors. In fact, by projecting heatmaps to latent space, we keep a lower number of dimensions, which are more representative of the meaningful directions of variability of the inputs, possibly avoiding the construction of degenerate clusters. For generating latent vectors, we choose the *t-SNE* algorithm [58, 147], since it projects similar inputs to neighbouring points and dissimilar inputs to distant points with high probability.

We generate latent vectors in two modes: (1) *global latent*: the projections of explanations in the latent space are computed considering inputs from all output classes; (2) *local latent*: the projections of explanations in the latent space are computed considering only inputs from a specific class. While the local latent space can achieve good separation of inputs from a specific class, the global latent space aims at separating both different classes of inputs and inputs within each class. The former is better if the generated inputs stay confined within one class, while the latter might be beneficial when the generated inputs tend to cross the borders between classes.

Clustering: The explanations generated by high-level techniques are FM cells based on high-level feature values of the inputs, whereas low-level explanations are vectors of contributions of low-level input elements/regions. Therefore, there is no way to directly measure the similarity between explanations at these two different abstraction levels. To make the comparison feasible, we propose to compare the *clusters of explanations* instead of the raw explanations (which are uncomparable). The underlying idea is the following: if two explanatory techniques group the inputs in a similar way, then they can be deemed similar; otherwise they are different.

For high-level explanations we use map cells as clusters. For low-level explanations, we need to group the explanations by using clustering approaches in such a way that objects in the same group, i.e., a cluster, are more similar to each other than to those in other groups. There exist multiple clustering techniques in the literature [93, 71, 29, 36]. For our study, we rely on the *Affinity Propagation* clustering technique [36] which recursively exchanges messages between data points; such messages encode the affinity of one point when choosing another point as its neighbor. The recursive exchange of messages continues until a set of highly-affine groups emerges. The main advantage of this clustering technique is that, unlike other techniques such as K-Means [93], (1) all the points are considered as possible centroids of the clusters, which avoids biasing the clusters to some randomly chosen points, used as the initial centroids; and (2) there is no need to provide the number of clusters to the algorithm in advance. In this step, we cluster the explanations in the three considered input spaces: *original*, consisting of raw explanation vectors; *global latent*, consisting of globally projected explanations; and *local latent*, consisting of locally projected explanations. Of course, each considered space may lead to different clusters, based on the distribution of the inputs in the corresponding space. The output of this step consists of the clusters generated for the fine-grained and the coarse-grained explanations, in each of the considered input spaces.

Comparison: In the previous steps, we generated high-level and low-level explanations for the considered inputs and processed them to generate clusters. The main evaluation step of our study is the comparison of these explanation clusters. Existing similarity or distance metrics to compare two sets of clusters, such as the MoJo metric [145], are based on the transformation of one cluster into the other. The computational complexity involved in the computation of these metrics is typically high (exponential in the worst case), but what is even more concerning in our usage scenario is that such metrics are highly sensitive to the number of clusters, and when there is a disparity in such number the distance tends to grow (similarity tends to decrease), because more transformation steps are needed to change one clustering into the other. However, a high disparity in the number of clusters is not necessarily an indicator of distance in our case. In fact, if all clusters in one set are *pure*, because their elements come from the same clusters in the other set, we deem the two clusterings very similar between each other, regardless of any disparity in the number of clusters. To capture such a notion of similarity between two sets of clusters, we define a novel metric based on *Gini Impurity (GI)* [121]. GI reflects the impurity level of a group of entities by indicating the probability that two samples from the given group have different labels, i.e., belong to different classes. So, to compute GI we need to define what are the *groups* and what are the *labels*. In our setting, when comparing two sets of clusters (low- vs high-level explanations), groups are the clusters identified in one set (source clustering), while labels are the cluster identifiers from the other set (target clustering). Hence, the impurity of a group of data D (i.e., a cluster from the source clustering) against the target clustering A can be measured as follows:

$$GI(D,\ A) = 1 - \sum_{i=1}^{|A|} p_{Ai}^2 \tag{8.1}$$

where $|A|$ is equal to the number of clusters in A, and p_{Ai} is the probability that cluster id i of clustering A occurs in dataset D. GI ranges between 0 and 1, being 0 when D contains no impurity (i.e., all its elements belong to the same cluster from A) and being minimum when D is uniformly impure (i.e., all its elements belong to a different cluster from A).

Let us consider cluster $CB1$ in Figure 8.2 (d). GI of $CB1$ against clustering A (Figure 8.2 (a)) is computed as follows: $GI(CB1, A) = 1 - \sum_{i=1}^{3} p_{Ai}^2 = 1 - (1+0+0) = 0$. ($p_{A1} = 1$ is the probability of cluster $CA1$ to be found in cluster $CB1$; $p_{A2} = p_{A3} = 0$ is the probability to find $CA2$, $CA3$).

We can now aggregate the computation of GI across all clusters that belong to the source clustering B, using the clusters in A as labels, by taking the average cluster impurity. The complement of such average impurity gives a *Gini Similarity* (GS) metric between clusterings, ranging between 0 and 1, where 1 is achieved when all clusters in B are pure (i.e., have GI = 0):

$$GS_{(B,A)} = 1 - \frac{1}{|B|} \sum_{i=1}^{|B|} GI(C_i, A) \tag{8.2}$$

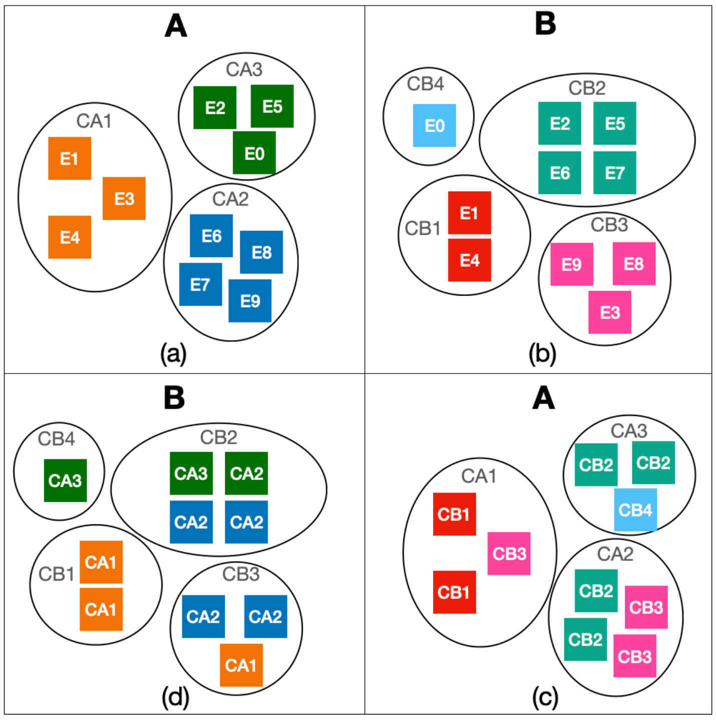


Figure 8.2. Gini Impurity (GI) computation with source B and target A (col. (d), (a)), and with source A and target B (col. (c), (b)); colors and text in (d), (c) are used to indicate the labels obtained from the target clusters (resp. (a), (b))

where $|B|$ is equal to the number of clusters in clustering B, and C_i is the i^{th} cluster in clustering B.

Figure 8.2 (a) and (b) present two examples of clusterings A and B. To compute the similarity between A and B, we consider the clustering B as source and then we color (and label) its elements based on the clusters in clustering A (see Figure 8.2 (d)). $GS_{(B,A)} = 1-\frac{1}{4}(GI(CB1,A)+GI(CB2,A)+GI(CB3,A)+GI(CB4,A)) = 1-\frac{1}{4}(0+\frac{1}{2}+\frac{4}{9}+0) = 0.76$.

GS is not symmetric by definition, as the labels assigned to the elements of one clustering depend on the other, and such labeling changes when we swap the two clusterings in the computation. For the example in Figure 8.2, let us now compute the similarity between A and B. We now consider the clustering A as source and we color (label) its elements based on the clusters they belong to in clustering B (see Figure 8.2 (c)). $GS_{(A,B)} = 1-\frac{1}{3}(GI(CA1,B)+GI(CA2,B)+GI(CA3,B)) = 1-\frac{1}{3}(\frac{4}{9}+\frac{1}{2}+\frac{4}{9}) = 0.53$.

To deal with the asymmetric nature of our metric, we report the maximum of the two GS values obtained when considering source and target clusterings in both orders:

$$Similarity(A,B) = Max(GS_{(A,B)}, GS_{(B,A)}) \tag{8.3}$$

Considering clusters A and B in Figure 8.2, $Similarity(A,B) = Max(0.53, 0.76) =$

0.76

Human Assessment: In this step, we conduct a human study to investigate the understandability of explanations. We design a survey and provide human assessors with low- and high-level explanations of misbehaviour-inducing inputs. Then, we ask the assessors whether the shown explanations effectively indicate the cause of the misbehaviour.

8.2 Empirical Study

8.2.1 Research Questions

RQ1 (Similarity): *How similar are high-level and low-level explanations of DL misbehaviours?*

High- and low-level approaches provide explanations about failure-inducing inputs from different perspectives. We aim to measure the similarity of these different explanations to assess to what extent they are comparable or complementary.

Metrics: To assess how similar the explanations generated by high-level and low-level techniques are, we compare the way inputs are grouped by similar explanations. To this aim, we measure (1) the number of clusters generated by each technique and (2) the similarity between these clusters according to our custom Gini Similarity (GS) metric. If two techniques produce explanations that group the inputs in a similar way (i.e., they produce nearly the same number of clusters that are pairwise very similar to each other), then they can be deemed similar; otherwise they differ, as they partition the input vectors in a different way.

RQ2 (Understandability): *How understandable are high-level and low-level explanations of DL misbehaviours?*

Since high-level techniques quantify high-level input features, while low-level approaches highlight low-level elements of the inputs, it is important to investigate whether these two types of explanations are equally understandable to humans.

Metrics: Effectively assessing the understandability of an explanation requires humans in the loop. Therefore, we designed a human study to assess if explanations are understandable and if they match the human expectations. We count the number of cases in which the explanation Matches with Human (MH), i.e., the number of times human assessors select a given explanation as possibly pointing to the cause of the misbehaviour.

8.2.2 Experimental Procedure

In our study, we consider two DL systems belonging to different domains, i.e., handwritten digit recognition and sentiment analysis, which have been widely used in the literature to generate explanations for DL systems [115, 139].

Table 8.1. Hyperparameters and configuration details

	MNIST	IMDB
Number of cells per feature	5	5
Number of runs	10	10
t-SNE components	2	2
t-SNE perplexity	1	1
Similarity metric	Euclidean	Euclidean
Input size	28×28	2000
Vocabulary size	-	10000
Number of inputs	250	250
Target class label	“5”	“positive”
Number of steps for IG	50	50
Batch size for IG	64	100
Number of samples for LIME	100	5000
Explanation library	*Xplique*	*Alibi*, LIME

To obtain the explanations, we adopted the pipeline described in Section 8.1 (configuration reported in Table 8.1).

As misbehaviour-inducing inputs, for MNIST, we used a set of 250 inputs belonging to the same ground-truth class (i.e., digit "5"), either from the test set or generated by test input generators. We used the DeepJanus [119] and Sinvad [66] test generators since they (1) belong to different families of approaches (i.e., model-based and DL generative, respectively), and (2) have been demonstrated to produce the highest ratio of valid input that preserve their ground-truth label [120]. In particular, we could obtain only 11 misclassified inputs belonging to the class digit "5" from the MNIST test set, due to high accuracy of the considered model. Therefore, we also included 35 inputs generated by DeepJanus and 204 inputs generated by Sinvad, to have enough and diverse misbehaviour-inducing inputs for our study.

For IMDB, we also used a set of 250 inputs either from the test set or generated by test input generators, all belonging to the same class ("positive" sentiment). We obtained 1924 misclassified inputs belonging to the class "positive" from the test set. Then, to have more diverse inputs we also included 663 inputs generated by DeepHyperion-CS (Chapter 5). Finally, we randomly selected 250 inputs from the overall set of 2587 candidate inputs.

To obtain low-level explanations, we used three open source libraries for images and textual inputs, i.e., *Xqlique* [32] for image, and *Alibi* [74] and LIME [116] for text.

To obtain high-level explanations, we needed high-level features for each considered domain. For MNIST inputs, we used the three high-level features (Section 5.1.3) defined in our study involving human experts, i.e., (1) *Luminosity (Lum)*: number of light pixels in the image, obtained by counting the pixels whose value is above 127; (2)

Orientation (Or): vertical orientation of the digit, obtained by computing the angular coefficient of the linear regression of the non-black pixels; (3) *Moves* (Mov): sum of the Euclidean distances between pairs of consecutive segments of the digit, the distance being zero when consecutive segments are connected and greater than zero when there are discontinuities. For IMDB reviews, we defined three features: (1) *Positive word count (Pos)*: number of words in the text with positive polarity, obtained by counting the words tagged as positive in the English Opinion Lexicon [59]; (2) *Negative word count (Neg)*: number of words in the text with negative polarity, obtained by counting the words tagged as negative in the English Opinion Lexicon; (3) *Verb count (Verb)*: number of verbs in the text, a proxy for the text complexity, computed by counting the words with a *verb* tag, according to the part-of-speech (POS) tagging produced by the NLTK library.

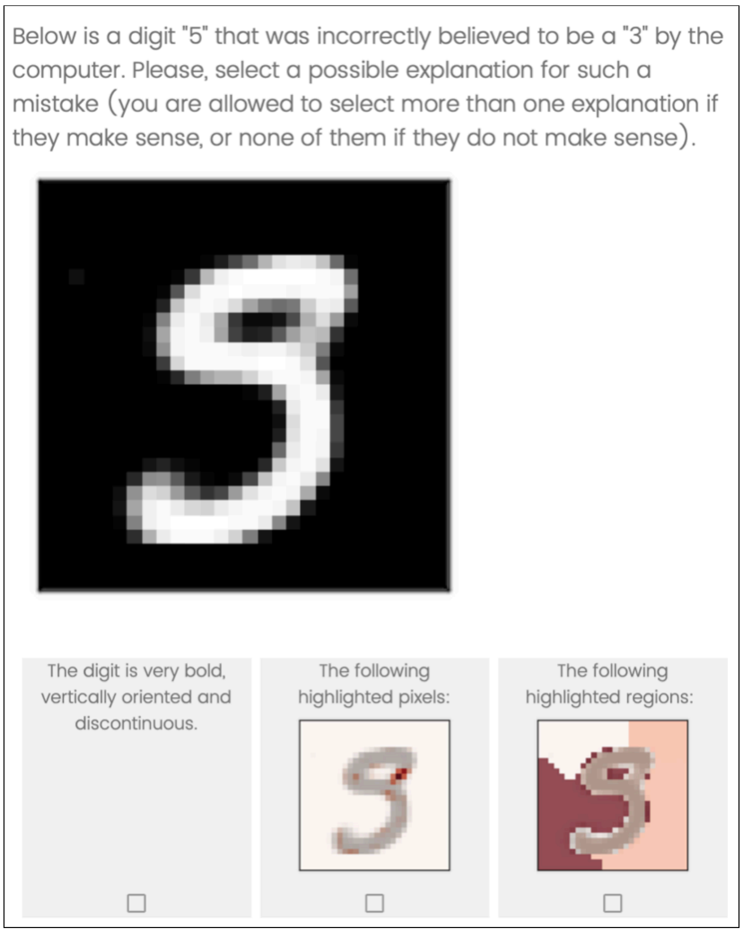
Our experimental procedure consists of the following two steps: (1) high- and low-level explanations comparison, using clustering; and (2) human assessment of the explanations, by means of two surveys: one for MNIST and one for IMDB.

High- and Low-Level Explanations Comparison

We generate Feature Maps with 1, 2, and 3 dimensions re-scaled to size 5 per dimension. We selected 5 as number of cells since the number of clusters (i.e., filled cells) with this configuration is comparable to the number of clusters obtained through low-level techniques, which allowed us to interpret each cell as a cluster without any need for an additional clustering step. We ran FM generation only once since the clusters (i.e., the cells) are deterministic.

As regards low-level explanations, we ran IG and LIME approaches 10 times each, since the corresponding clusterings are obtained trough t-SNE, which is non-deterministic. To obtain the best configuration for the *t-SNE* hyperparameters (i.e., number of components and perplexity), we performed preliminary runs with different configurations and selected the one producing the highest *silhouette score*. The best configurations of the hyperparameters of IG and LIME are reported in Table 8.1.

For clustering the low-level explanations, we used the *Euclidean distance* to compute the similarity matrices, where distances are computed in three different vector spaces: *original*, *global latent*, and *local latent* (see Section 8.1). In the original space, we consider heatmaps as vectors to be clustered. For MNIST, a heatmap is a 28×28 matrix, where each vector component e_{ij} is the contribution value of the pixel occupying the i-th row and the j-th column in the image. The matrix is flattened into a vector before applying clustering. For IMDB, a heatmap is a vector of size 10000, the size of the vocabulary used by the tokeniser (which maps each word to the corresponding one-hot encoding) and each vector component e_i is the contribution value of the i-th word in the text. In both global latent and local latent spaces, we use vectors of size 2, the same number of dimensions that was found to be optimal for the t-SNE algorithm.


Below is a digit "5" that was incorrectly believed to be a "3" by the computer. Please, select a possible explanation for such a mistake (you are allowed to select more than one explanation if they make sense, or none of them if they do not make sense).

The digit is very bold, vertically oriented and discontinuous. ☐

The following highlighted pixels: ☐

The following highlighted regions: ☐

Figure 8.3. Sample survey question for MNIST

Human Assessment of the Explanations

To determine whether an explanation is human-understandable, we asked humans to determine which of the explanations provided them meaningful information about the reason why the model misbehaves for specific inputs. We published two surveys (one for each case study) by using Qualtrics, a survey platform commonly used also for software engineering research [148, 88]. To ensure the assessment quality, we selected software engineering researchers and allowed each of them to devote up to 1 week to answer the questions and take at most 1 questionnaire for each case study.

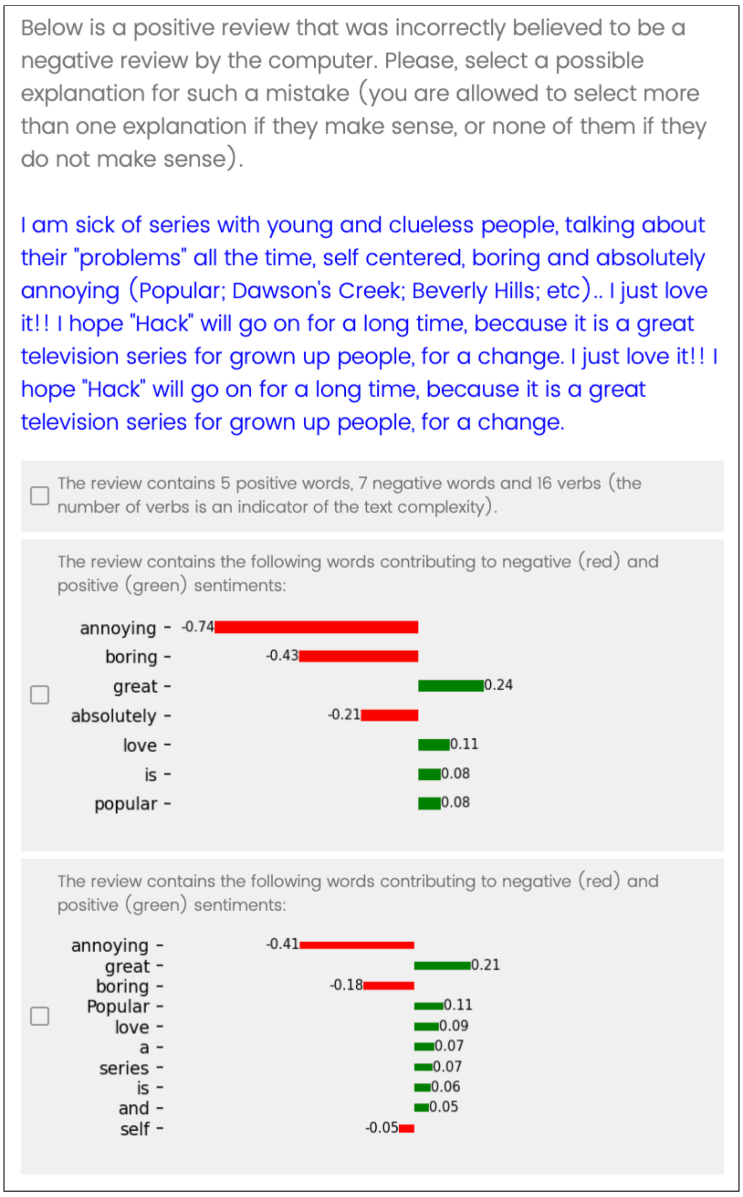
For MNIST, we created a survey with 10 questions to be answered by human assessors. To this aim, we randomly selected 10 "5" digit images from the MNIST data set which are misclassified by the considered model and we computed high-level and low-level explanations for each of them. More specifically, for the high-level explanations we reported the feature values provided by the three-dimensional FM and for the low-level

Table 8.2. RQ1 - Number of clusters (NC) for high-level techniques with different feature combinations for MNIST and IMDB.

	MNIST		IMDB	
HL Technique	Features	NC	Features	NC
Feature map 3D	Mov-Lum-Or	31	Pos-Neg-Verb	27
	Mov-Lum	17	Pos-Neg	15
Feature map 2D	Lum-Or	15	Neg-Verb	14
	Or-Mov	14	Pos-Verb	12
	Mov	5	Pos	5
Feature map 1D	Lum	5	Neg	5
	Or	5	Verb	5

explanations we visualized IG and LIME heatmaps, overlayed on the original digit images. We used 3D FEATURE MAPS for the human study to include all the available features for the high-level explanations. For each MNIST image, we showed the human assessors the possible explanations (i.e. FM, IG, and LIME) and asked them to answer the following question: "*Below is a digit "5" that was incorrectly believed to be another digit by the computer. Please, select a possible explanation for such a mistake (you are allowed to select more than one explanation if they make sense, or none of them if they do not make sense)*". Figure 8.3 shows a misclassified digit "5" with the three different explanations. The assessors could select the explanations that they considered as a plausible reason for the misbehaviour. In particular, the assessors were allowed to choose zero or more explanations. We collected 29 answers from the human assessors in the MNIST survey.

For IMDB, we created a survey with 10 questions by randomly selecting 10 positive reviews from the IMDB dataset, which are misclassified by the model. For the high-level explanations, we reported the feature values provided by the three-dimensional FM and the explanations generated by LIME and IG and we visualised the words with highest contributions to negative and positive sentiments in a bar chart (i.e., we reported up to 10 most contributing words with non-zero contribution). For each text, we asked humans to answer the following question: "*Below is a positive review that was incorrectly believed to be a negative review by the computer. Please, select a possible explanation for such a mistake (you are allowed to select more than one explanations if they make sense, or none of them if they do not make sense)*". Also in this case, we showed explanations provided by the FM, IG, and LIME (as shown in Figure 8.4). In the IMDB survey we collected 19 answers from the human assessors.



Below is a positive review that was incorrectly believed to be a negative review by the computer. Please, select a possible explanation for such a mistake (you are allowed to select more than one explanation if they make sense, or none of them if they do not make sense).

I am sick of series with young and clueless people, talking about their "problems" all the time, self centered, boring and absolutely annoying (Popular; Dawson's Creek; Beverly Hills; etc).. I just love it!! I hope "Hack" will go on for a long time, because it is a great television series for grown up people, for a change. I just love it!! I hope "Hack" will go on for a long time, because it is a great television series for grown up people, for a change.

☐ The review contains 5 positive words, 7 negative words and 16 verbs (the number of verbs is an indicator of the text complexity).

☐ The review contains the following words contributing to negative (red) and positive (green) sentiments:

☐ The review contains the following words contributing to negative (red) and positive (green) sentiments:

Figure 8.4. Sample survey question for IMDB

8.2.3 Results

RQ1: Similarity

Tables 8.2 and 8.3 report the number of clusters generated by each explanatory technique. For each low-level approach, we report the minimum and maximum number of clusters (NC) in order to show their variability due to non-determinism.

For MNIST, the number of clusters generated by FM 3D is 31, larger than the

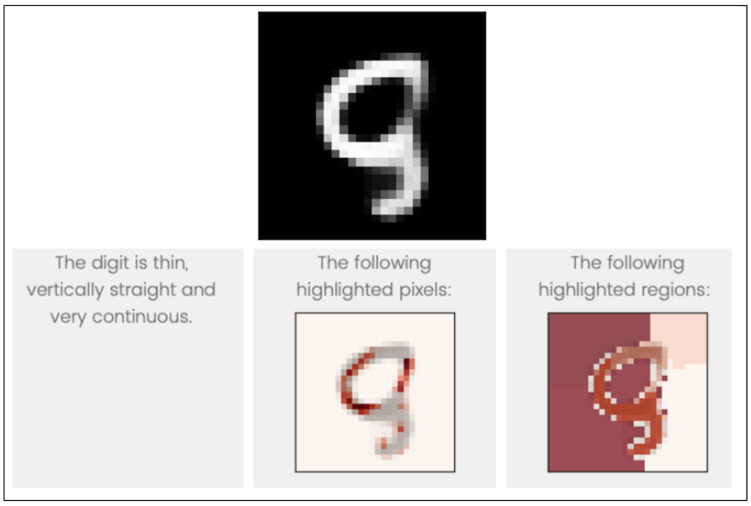
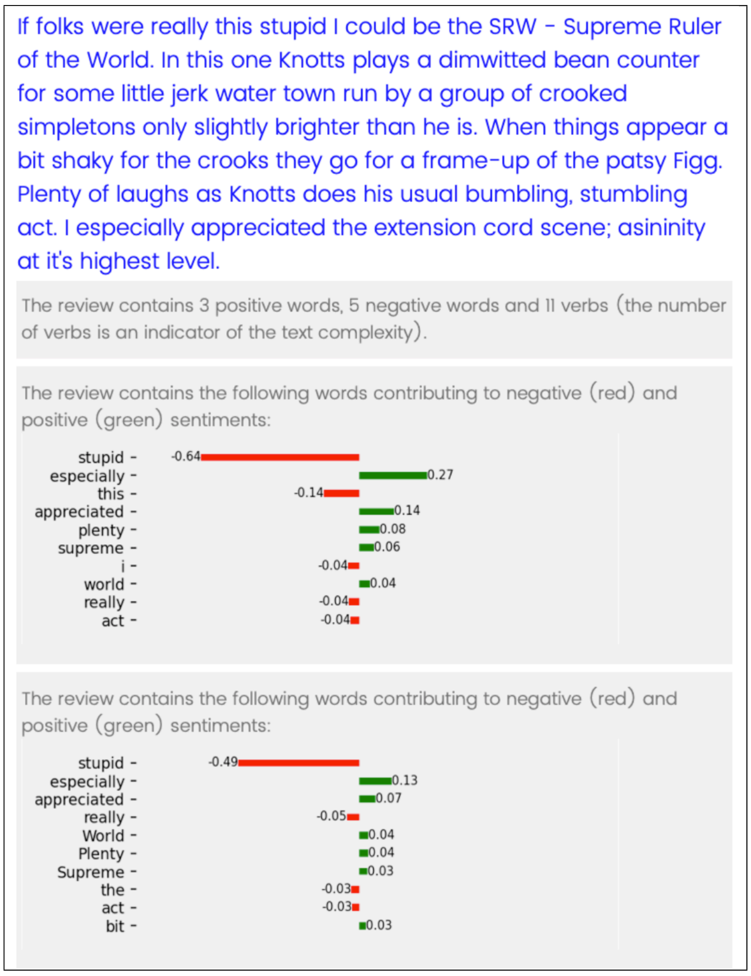

The digit is thin, vertically straight and very continuous.
The following highlighted pixels:
The following highlighted regions:

Figure 8.5. MNIST input from the human study for which the majority selected IG.

Table 8.3. RQ1 - Number of clusters (NC) for low-level techniques for MNIST and IMDB.

		MNIST	IMDB
LL Technique	Input space	NC [min, max]	NC [min, max]
IG	Original	44.0 [44, 44]	35.1 [35, 36]
	Global Latent	18.7 [17, 20]	14.8 [12, 18]
	Local Latent	20.5 [19, 21]	18.2 [16, 22]
LIME	Original	39 [37, 44]	38.5 [37, 39]
	Global Latent	17.7 [16, 19]	17.3 [16, 19]
	Local Latent	19.0 [17, 21]	18.3 [16, 21]



If folks were really this stupid I could be the SRW - Supreme Ruler of the World. In this one Knotts plays a dimwitted bean counter for some little jerk water town run by a group of crooked simpletons only slightly brighter than he is. When things appear a bit shaky for the crooks they go for a frame-up of the patsy Figg. Plenty of laughs as Knotts does his usual bumbling, stumbling act. I especially appreciated the extension cord scene; asininity at it's highest level.

The review contains 3 positive words, 5 negative words and 11 verbs (the number of verbs is an indicator of the text complexity).

The review contains the following words contributing to negative (red) and positive (green) sentiments:

The review contains the following words contributing to negative (red) and positive (green) sentiments:

Figure 8.6. IMDB input from the human study for which the majority selected both FM and IG explanations.

number of clusters generated by FM with any other dimensionality (i.e., 1D and 2D). Not surprisingly, the lower the dimensionality, the lower the number of clusters. In fact, all the FM dimensions were rescaled to size 5, leading to a lower number of cells for lower dimensionality maps (e.g., in 3D maps there are at most 125 cells to fill, while for 1D maps there are only 5 cells). The highest difference between high- and low-level techniques is between FM 1D and IG in the Original input space (39 clusters). Instead, the NC value for both FM 2D and 3D is comparable to the number of clusters generated by low-level techniques (resp. global/local latent space and original space).

As regards IMDB, the highest difference between high- and low-level techniques is the one between FM 1D and LIME in the Original input space, i.e., 33.5. Also for IMDB, the closest NC values between the two classes of techniques is produced by FM 2D and low-level techniques in the global/local latent space. The number of clusters for FM 3D is comparable to low-level techniques in the original space.

For both case studies, IG produces a more stable number of clusters in the original space due to its deterministic nature (i.e. it is based on a mathematical equation which computes linear interpolation from a baseline), whereas there are some variations for global and local latent spaces due to the non-deterministic nature of the *t-SNE* algorithm. Instead, LIME always generates slightly different explanations across multiple runs, because of the randomness introduced by input sampling and surrogate model training during the explanation computation, and thus produces a variable number of clusters.

Table 8.4 reports the similarity between clusterings. To assess the statistical significance of the comparisons between different configurations, we performed the Mann-Whitney U-test and measured the effect size by means of the Vargha-Delaney's A12 statistic [5]. For both MNIST and IMDB (fifth and seventh columns), the similarity values between high- and low-level techniques are almost always significantly higher (p-value < 0.05, large effect size) when considering the original input space, rather than global and local latent spaces within the same configuration (i.e., up to 0.34 more in the comparison between FM 1D and IG for MNIST). This result may be due to the fact that the heatmaps in the original space retain more feature-related information than their projections into the latent space. For MNIST, the highest similarity is achieved between FM 1D (Orientation) and IG in the original space (0.94). For IMDB, the highest clustering similarity is between FM that considers the Verb feature only and LIME explanations in the original space (0.89). The most different partitions between low- and high-level techniques (i.e., lowest similarity) are obtained when considering FM 2D and low-level explanations in the latent space (either local or global). In fact, for MNIST we have the lowest similarity when considering Mov-Lum maps and LIME in the global latent space (0.39), whereas for IMDB the lowest similarity is between Pos-Neg maps and IG in the global latent space.

On average, the similarity between FM and IG, across input spaces and subjects, is 0.65, while it is 0.64 with LIME. These are relatively low values. As a reference, such similarity values are obtained when a cluster with 100 elements contains 23 impure

Table 8.4. RQ1 - Comparing high-level and low-level explanations' Similarity (Sim), considering different input spaces for MNIST and IMDB; boldface indicate statistical significance when comparing original with latent space similarities.

			MNIST		IMDB	
HL technique	LL technique	Input space	Features	Sim	Features	Sim
Feature map 3D	IG	Original	Mov-Lum-Or	**0.70**	Pos-Neg-Verb	**0.74**
		Global Latent		0.55		0.68
		Local Latent		0.55		0.66
	LIME	Original		0.55		**0.81**
		Global Latent		0.53		0.66
		Local Latent		0.53		0.68
Feature map 2D	IG	Original	Mov-Lum	**0.71**	Pos-Neg	**0.74**
		Global Latent		0.40		0.52
		Local Latent		0.41		0.53
	LIME	Original		**0.56**		**0.77**
		Global Latent		0.39		0.55
		Local Latent		0.40		0.56
Feature map 2D	IG	Original	Lum-or	**0.76**	Neg-Verb	**0.78**
		Global Latent		0.52		0.60
		Local Latent		0.52		0.60
	LIME	Original		**0.65**		**0.82**
		Global Latent		0.49		0.61
		Local Latent		0.49		0.61
Feature map 2D	IG	Original	Mov-Or	**0.80**	Pos-Verb	**0.77**
		Global Latent		0.54		0.57
		Local Latent		0.54		0.59
	LIME	Original		**0.64**		**0.81**
		Global Latent		0.52		0.61
		Local Latent		0.51		0.63
Feature map 1D	IG	Original	Mov	**0.83**	Pos	**0.80**
		Global Latent		0.49		0.64
		Local Latent		0.52		0.65
	LIME	Original		**0.68**		**0.83**
		Global Latent		0.50		0.66
		Local Latent		0.54		0.69
Feature map 1D	IG	Original	Lum	**0.78**	Neg	**0.82**
		Global Latent		0.47		0.63
		Local Latent		0.53		0.65
	LIME	Original		**0.69**		**0.84**
		Global Latent		0.45		0.67
		Local Latent		0.50		0.67
Feature map 1D	IG	Original	Or	**0.94**	Verb	**0.87**
		Global Latent		0.82		0.71
		Local Latent		0.83		0.73
	LIME	Original		**0.89**		**0.89**
		Global Latent		0.82		0.72
		Local Latent		0.83		0.74

Table 8.5. RQ2 - Number of Matches with Human Explanations (MH); 'None' indicates the number of cases when no match was found.

	MH							
	MNIST				IMDB			
Q#	FM 3D	IG	LIME	None	FM 3D	IG	LIME	None
Q1	12	2	10	8	2	13	3	3
Q2	5	23	5	1	5	3	3	9
Q3	4	7	9	12	11	17	8	0
Q4	6	7	5	14	6	15	3	2
Q5	3	15	2	11	10	14	7	1
Q6	7	6	4	14	12	15	8	0
Q7	7	11	7	10	0	14	5	3
Q8	9	5	8	13	11	14	12	1
Q9	5	1	9	17	6	8	4	4
Q10	13	10	5	8	11	12	7	2
Sum	71	87	64	108	74	125	60	25

elements, i.e., 23 elements that are assigned a different cluster by the other technique (see Equation 8.2).

RQ1: High-level and low-level techniques partition inputs in different ways. Despite FM 2D and low-level techniques in the global latent space produce nearly the same number of clusters, those show low similarity. Likewise, FM 1D produces clusterings similar to the ones obtained by low-level techniques in the original space, but with very different number of clusters.

RQ2: Understandability

Table 8.5 shows the results extracted from questionnaires for MNIST and IMDB. Each column reports the number of assessors who chose the explanation provided by each considered technique (choices were not mutually exclusive). The last column highlights the cases for which no explanation was selected by the assessor. Since each assessor can select 0 to 3 explanations in each question, the sum of the values in each row does not correspond to the number of assessors.

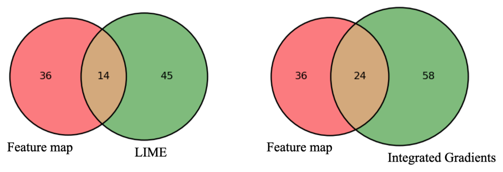
Table 8.5 (left) reports the answers we collected for MNIST. IG was selected more than other explanations 4 times, e.g., for Q2 IG has been chosen 18 times more than the others. The explanations by FM are selected more than the other explanations 4 times. LIME was selected more than the others in only 2 questions. We can conclude that there is no technique clearly more understandable than the others for explaining digit misclassications, according to human assessors. Overall, the explanations generated by IG match more frequently with the human expectations, as they have been selected 30% of the times by the assessors. FM explanations and LIME heatmaps have been selected 24% and 22% of the times. However, in more than one third of the answers the assessors chose none of the explanations. This result suggests that there is large room for improvement in explaining image misclassifications. Table 8.5 (right) reports the answers for IMDB. IG was selected more than the other techniques 9 times, while FM was selected more than the others in the remaining case (i.e., Q2). Instead, LIME explanations were never selected more often than the other techniques. IG was selected the highest number of times by human assessors, i.e., 66% , followed by FM with 39% and LIME with 31%. Explanations were more understandable for textual inputs, than for handwritten digit images. In fact, only 13% of the times the assessors did not choose any explanation for a question on IMDB. For MNIST, only for 2 questions we have a majority, i.e., more than half of the assessors selected the same answer. Instead, for IMDB in 8 questions we have answers selected by the majority of assessors. This indicates that the explanations provided for IMDB find more consensus among humans than the explanations provided for MNIST.

RQ2: Both high- and low-level techniques produce human-understandable explanations of misbehaviours of text classifiers. In particular, IG explanations were selected 125 times out of 190 answers for IMDB. On the other hand, the explanations for misbehaviours of image classifiers poorly matched with the human judgement. In fact, in more than one third of the answers the assessors chose none of the proposed explanations for MNIST. On the remaining two thirds, high- and low-level explanations are chosen approximately the same number of times.

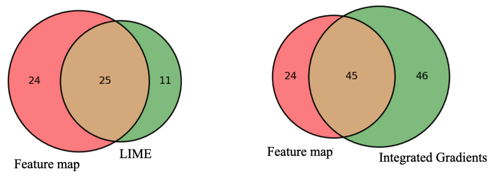
Discussion

The answers provided by the human assessors offer several insights that we discuss qualitatively in the following.

- For MNIST, the two most understandable explanations are IG and FM. Not only are these explanations at different levels (i.e., low- and high-level, respectively), but they show also some degree of complementarity. In fact, for Q1, Q8 and Q9, FM were selected more times than IG with a MH difference higher than 4. On the other hand, for Q2, Q5 and Q7, IG was selected more times than FM. Remarkably, for question Q2 (reported in Figure 8.5), IG was selected 18 times more than FM. In this case, the assessors did not find the FM explanation useful, while IG highlights the pixels that make the upper part of the five look like a nine, i.e., the leftmost pixels, making the upper part round, and the rightmost pixels, closing the circle.



(a) MNIST



(b) IMDB

Figure 8.7. Comparison of the number of matches with human between high-level and low-level techniques.

- For IMDB, FM and IG explanations are the most chosen by human assessors. In particular, for half of the questions, IG and FM were chosen by at least 10 assessors and the difference in MH was lower than or equal to 4. For instance, Figure 8.6 reports Q8, where the assessors selected FM 11 times and IG 14 times. In this case, the assessors acknowledged that the review actually contained more negative than positive words as reported by the FM explanation. At the same time, the assessors probably agreed with the IG heatmap that the word "stupid" contained in the text can negatively affect the prediction.

- For some questions, all the explanations were judged as poorly understandable by humans, who did not choose any of the provided explanations as possible causes of the misbehaviour. For MNIST, Q9 can be considered the most challenging question, since only 15 times out of 87 an explanation was selected, while the majority of the assessors selected none of the explanations. Similarly, in the IMDB survey, Q2 is the question with the lowest number of selected explanations, since the assessors chose an explanation only 11 times out of 57, while the majority chose none.

- As shown in Figure 8.7 (a), for MNIST, assessors selected only FM 36 times and only LIME 45 times while they selected both explanations 14 times. Similarly, they selected only FM 36 times, only IG 58 times and both FM and IG 24 times. This suggests that high-level and low-level explanations for MNIST digit misclassifications are highly complementary. Figure 8.7 (b) shows that for IMDB, most of the times both high- and low-level explanations have been selected together. However, the number of times assessors selected one of the two and not the other remains quite high, which indicates substantial complementarity.

In summary, in several instances either high- or low-level explanations look understandable for humans and provide hints on what are the properties of the inputs that make the DNN misbehave. In most cases, either high- or low-level explanations, but not both, were chosen as useful explanations, indicating that such explanations are highly complementary. On the other hand, there is still a large fraction of cases in which none of the two are deemed useful by humans, which points to the need for better DNN explanations.

Threats to Validity

Construct Validity: FM depends on the features selected to generate the map dimensions. For feature selection, we relied on previous work [159], while for the combination of features, we exhaustively considered all available dimensions and all their combinations. Moreover, the selection of different hyper-parameters for our study can affect the final results. To reduce this risk, we developed our framework in order to be configurable and we performed some preliminary runs with different hyper-parameters to select the best configuration.

External Validity: The selection of subjects could be a threat to external validity. Therefore, we considered two different domains, i.e., image and text. To represent different types of explanations, we considered one state-of-the-art, openly available technique for each explanation type.

Conclusion Validity: The stochastic nature of some of the considered low-level techniques, the dimensionality reduction approach and the clustering algorithm might affect the final outcome. To mitigate this threat, we ran these techniques multiple times, reported average value across runs, and conducted statistical tests.

8.3 Conclusion

Due to the growing adoption of DL software in solving complex and safety critical tasks, understanding the behaviour of DL systems attracted enormous attention in the research community. As the demand for explainable DL models rises, a large variety of techniques have been proposed to interpret the DL system's behaviour.

We provided an extensive and in-depth analysis and comparison of explanatory techniques with different granularity levels. To this aim, we adopt a novel methodology to compare the different explanatory techniques proposed in the literature.

Our empirical results show that while high-level and low-level explanation are both understandable for humans, they provide different and complementary insights about failure-inducing inputs. Furthermore, our human study suggests that current explanations are not always satisfactory, as they do not provide human-interpretable causes of misbehaviours.

8.4 Reproducibility

The code implementing our comparison method, the dataset, and all the scripts to replicate the experimental evaluation are available online [164].

Chapter 9

Conclusion

In this thesis, we explored the topic of exposing and explaining misbehaviours of DL systems. We resorted to Illumination Search to find the highest-performing test cases (i.e., misbehaving and closest to misbehaving), spread across the cells of a map representing the feature space of the system. We introduced a methodology that guides the users of our approach in the tasks of identifying and quantifying the dimensions of the feature space for a given domain. We developed DEEPHYPERION-CS, a search-based tool for DL systems that illuminates, i.e., explores at large, the feature space, by providing developers with an interpretable feature map where automatically generated inputs are placed along with information about the exposed behaviours. Our results showed that DEEPHYPERION-CS outperforms state-of-the-art tools. It exposed significantly more misbehaviours for 5 out of 6 feature combinations. DEEPHYPERION-CS was useful for expanding the datasets used to train the DL systems, populating up to 200% more feature map cells than the original training set.

When deployed in the operation environment, Deep Learning (DL) systems often experience the so-called development to operation (dev2op) data shift, which causes a lower prediction accuracy on field data as compared to the one measured on the test set during development. To address the dev2op shift, developers must obtain new data with the newly observed features, as these are under-represented in the train/test set, and use them to fine tune the DL model, so as to reach the desired accuracy level. We addressed the issue of acquiring new data with the specific features observed in operation, which caused a dev2op shift, by proposing DEEPATASH, a novel search-based focused testing approach for DL systems. DEEPATASH targets a cell in the feature space, defined as a combination of feature ranges, to generate misbehaviour-inducing inputs with predefined features. Experimental results showed that DEEPATASH was able to generate up to 29× more targeted, failure-inducing inputs than the baseline approach. The inputs generated by DEEPATASH were useful to significantly improve the quality of the original DL systems through fine tuning not only on data with the targeted features, but quite surprisingly also on inputs drawn from the original distribution.

The resource-intensive nature of testing ADSs requires efficient methodologies for generating targeted and diverse tests. We introduced a novel approach, DEEPATASH-LR, that incorporates a surrogate model into the focused test generation process. This integration significantly improves focused testing effectiveness and applicability in resource-intensive scenarios. Experimental results showed that the integration of the surrogate model is fundamental to the success of DEEPATASH-LR. Our approach was able to generate an average of up to 60× more targeted, failure-inducing inputs compared to the baseline approach. Illumination-based techniques require the expensive involvement of human experts in defining features of interest and metrics for their measurement. This limitation restricts the broader applicability of these testing approaches. We addressed this limitation with DEEPTHEIA, our fully automated illumination-based test generator that autonomously extracts features and explores the feature space using cutting-edge diffusion models. Experimental results showed that DEEPTHEIA consistently extracts highly discriminative features. Independent human assessors certified that DEEPTHEIA is able to group misbehaviour-inducing inputs in a way that is understandable to humans in over 78% of the cases.

With the increasing demand for explainable Deep Learning (DL) models, a plethora of techniques have emerged to interpret the behavior of DL systems. We undertook an extensive and thorough comparison of explanatory techniques, focusing on various granularity levels. Employing a novel methodology, we systematically compared different explanatory techniques proposed in the literature. Our empirical findings reveal that both high-level and low-level explanations are comprehensible to humans, offering distinct and complementary insights into failure-inducing inputs. However, our human study suggests that current explanations often fall short of providing satisfactory human-interpretable causes of misbehaviours.

9.1 Impact

The Feature maps introduced by our approach have been used for different testing tasks. Nguyen et al. [105] used feature map for test selection. In particular, they ensure the diversity between test inputs by selecting those that occupy different feature map cells.

Our feature maps have also been used to assess the adequacy of a test suite, measured as the number of feature cells covered by the test inputs in the test suite. This was done in a recent search-based testing competition to compare different test generators for ADSs [39, 13].

Biagiola et al. [14] used our feature maps to group failures by similarity and compare the performance of different simulators for testing autonomous driving systems. More specifically, they adopt DEEPHYPERION for test generation using two general-purpose, cheap simulators (aka, digital siblings). They merged the generated feature maps to combine the testing output of the two digital siblings and approximate the behavior of the model in the high-fidelity, expensive digital twin.

9.2 Future Work

Despite the remarkable results achieved so far, our work on testing DL systems using feature maps and illumination search is at its dawn and opens many interesting directions for future research. We plan to apply the feature maps produced by our approach to other DL software development tasks. In fact, we believe they are an intuitive approach for evaluating test set adequacy or guiding test selection.

Additionally, we plan to extend the application domain of our approach to a broader range of DL systems. Our preliminary results in the SBFT-UAV competition [69] showed that DeepHyperion-CS is effective in generating failure-inducing inputs associated with highly diverse features for Unmanned Aerial Vehicles (UAV). As a future work, we plan to define more effective features and extend the applicability of DeepHyperion-CS for UAV systems.

We are also interested in improving our proposed automatic image generation approach for complex systems considering domain-specific prompts on diffusion models.

Bibliography

[1] Abdessalem, R. B., Nejati, S., Briand, L. C. and Stifter, T. [2016]. Testing advanced driver assistance systems using multi-objective search and neural networks, *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, ACM, pp. 63–74.
URL: *https://doi.org/10.1145/2970276.2970311*

[2] Abdessalem, R. B., Nejati, S., Briand, L. C. and Stifter, T. [2018]. Testing vision-based control systems using learnable evolutionary algorithms, *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, ACM, pp. 1016–1026.
URL: *http://doi.acm.org/10.1145/3180155.3180160*

[3] Abdessalem, R. B., Panichella, A., Nejati, S., Briand, L. C. and Stifter, T. [2018]. Testing autonomous cars for feature interaction failures using many-objective search, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, ACM, pp. 143–154.
URL: *http://doi.acm.org/10.1145/3238147.3238192*

[4] Ahadit, A. B. and Jatoth, R. K. [2022]. A novel multi-feature fusion deep neural network using hog and vgg-face for facial expression classification, *Machine Vision and Applications* **33**(4): 55.

[5] Arcuri, A. and Briand, L. [2014]. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, *Software Testing, Verification and Reliability* **24**(3): 219–250.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1486*

[6] Arcuri, A. and Fraser, G. [2011]. On parameter tuning in search based software engineering, *in* M. B. Cohen and M. Ó. Cinnéide (eds), *Search Based Software Engineering - Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings*, Vol. 6956 of *Lecture Notes in Computer Science*, Springer, pp. 33–47.
URL: *https://doi.org/10.1007/978-3-642-23716-4_6*

[7] Arthur, D. and Vassilvitskii, S. [2006]. k-means++: The advantages of careful seeding, *Technical report*, Stanford.

[8] Attaoui, M., Fahmy, H., Pastore, F. and Briand, L. [2023]. Black-box safety analysis and retraining of dnns based on feature extraction and clustering, *ACM Transactions on Software Engineering and Methodology* **32**(3): 1–40.

[9] Back, T. [1994]. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms, *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 57–62 vol.1.

[10] BeamNG GmbH [2018]. BeamNG.research.
URL: *https://www.beamng.gmbh/research*

[11] Bellman, R. [1966]. Dynamic programming, *Science* **153**(3731): 34–37.
URL: *https://www.science.org/doi/abs/10.1126/science.153.3731.34*

[12] Bengio, Y. [2011]. Deep learning of representations for unsupervised and transfer learning, *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, JMLR.org, Washington, USA, p. 17â37.

[13] Biagiola, M., Klikovits, S., Peltomäki, J. and Riccio, V. [2023]. Sbft tool competition 2023-cyber-physical systems track, *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*, IEEE, pp. 45–48.

[14] Biagiola, M., Stocco, A., Riccio, V. and Tonella, P. [2023]. Two is better than one: Digital siblings to improve autonomous driving testing, *arXiv preprint arXiv:2305.08060* .

[15] Biagiola, M. and Tonella, P. [2023]. Testing of deep reinforcement learning agents with surrogate models, *arXiv preprint arXiv:2305.12751* .

[16] Biggio, B. and Roli, F. [2018]. Wild patterns: Ten years after the rise of adversarial machine learning, *Pattern Recognition* **84**: 317–331.

[17] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K. [2016]. End to end learning for self-driving cars, *CoRR* **abs/1604.07316**: 1–9.
URL: *http://arxiv.org/abs/1604.07316*

[18] Bouayed, A. M., Atif, K., Deriche, R. and Saim, A. [2020]. Improving auto-encoders' self-supervised image classification using pseudo-labelling via data augmentation and the perceptual loss.

[19] Carlini, N. and Wagner, D. [2017a]. Towards evaluating the robustness of neural networks, *2017 ieee symposium on security and privacy (sp)*, Ieee, pp. 39–57.

[20] Carlini, N. and Wagner, D. [2017b]. Towards evaluating the robustness of neural networks, *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 39–57.
URL: *https://doi.ieeecomputersociety.org/10.1109/SP.2017.49*

[21] Catmull, E. and Rom, R. [1974]. A class of local interpolating splines, *in* R. E. Barnhill and R. F. Riesenfeld (eds), *Computer Aided Geometric Design*, Academic Press, pp. 317 – 326.
URL: *http://www.sciencedirect.com/science/article/pii/B9780120790500500205*

[22] Chollet, F. [2020]. Simple mnist convnet, `https://github.com/keras-team/keras-io/blob/master/examples/vision/mnist_convnet.py`.

[23] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. [2002]. A fast and elitist multi-objective genetic algorithm: Nsga-ii, *IEEE transactions on evolutionary computation* **6**(2): 182–197.

[24] Demir, S., Eniser, H. F. and Sen, A. [2020]. Deepsmartfuzzer: Reward guided test generation for deep learning, *Proceedings of the Workshop on Artificial Intelligence Safety 2020 (IJCAI-PRICAI 2020), Yokohama, Japan, January, 2021*, Vol. 2640 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 134–140.
URL: *http://ceur-ws.org/Vol-2640/paper_19.pdf*

[25] Dhariwal, P. and Nichol, A. [2021]. Diffusion models beat gans on image synthesis, *Advances in neural information processing systems* **34**: 8780–8794.

[26] Dola, S., Dwyer, M. B. and Soffa, M. L. [2021]. Distribution-aware testing of neural networks using generative models, *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 226–237.

[27] Dola, S., Dwyer, M. B. and Soffa, M. L. [2023]. Input distribution coverage: Measuring feature interaction adequacy in neural network testing, *ACM Transactions on Software Engineering and Methodology* **32**(3): 1–48.

[28] Dunn, I., Pouget, H., Kroening, D. and Melham, T. [2021]. Exposing previously undetectable faults in deep neural networks, *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2021, Association for Computing Machinery, New York, NY, USA, pp. 56–66.
URL: *https://doi.org/10.1145/3460319.3464801*

[29] Ester, M., Kriegel, H.-P., Sander, J. and Xu, X. [1996]. A density-based algorithm for discovering clusters in large spatial databases with noise, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, AAAI Press, p. 226â231.

[30] Fahmy, H., Pastore, F., Bagherzadeh, M. and Briand, L. [2021]. Supporting deep neural network safety analysis and retraining through heatmap-based unsupervised learning, *IEEE Transactions on Reliability* **70**(4): 1641–1657.

[31] Fahmy, H., Pastore, F., Briand, L. and Stifter, T. [2022]. Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems, *arXiv preprint arXiv:2204.00480* .

[32] Fel, T., Hervier, L., Vigouroux, D., Poche, A., Plakoo, J., Cadene, R., Chalvidal, M., Colin, J., Boissin, T., Bethune, L., Picard, A., Nicodeme, C., Gardes, L., Flandin, G. and Serre, T. [2022]. Xplique: A deep learning explainability toolbox, *Workshop on Explainable Artificial Intelligence for Computer Vision (CVPR)* .

[33] Fraser, G. and Arcuri, A. [2011]. Evolutionary generation of whole test suites, *2011 11th International Conference on Quality Software*, IEEE, pp. 31–40.

[34] Fraser, G. and Arcuri, A. [2013]. Whole test suite generation, *IEEE Transactions on Software Engineering* **39**(2): 276–291.

[35] Fraser, G., Arcuri, A. and McMinn, P. [2015]. A memetic algorithm for whole test suite generation, *Journal of Systems and Software* **103**: 311–327.
URL: *https://www.sciencedirect.com/science/article/pii/S0164121214001216*

[36] Frey, B. J. and Dueck, D. [2007]. Clustering by passing messages between data points, *science* **315**(5814): 972–976.

[37] Fukunaga, K. [2013]. *Introduction to statistical pattern recognition*, Elsevier.

[38] Gal, R., Alaluf, Y., Atzmon, Y., Patashnik, O., Bermano, A. H., Chechik, G. and Cohen-Or, D. [2022]. An image is worth one word: Personalizing text-to-image generation using textual inversion, *arXiv preprint arXiv:2208.01618* .

[39] Gambi, A., Jahangirova, G., Riccio, V. and Zampetti, F. [2022]. Sbst tool competition 2022, *Proceedings of the 15th Workshop on Search-Based Software Testing*, pp. 25–32.

[40] Gambi, A., Müller, M. and Fraser, G. [2019]. Automatically testing self-driving cars with search-based procedural content generation, *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, ACM, pp. 318–328.
URL: *https://doi.org/10.1145/3293882.3330566*

[41] Goldberg, D. E. and Deb, K. [1991]. A comparative analysis of selection schemes used in genetic algorithms, Vol. 1 of *Foundations of Genetic Algorithms*, Elsevier, pp. 69–93.
URL: *https://www.sciencedirect.com/science/article/pii/B9780080506845500082*

[42] Goodfellow, I. J., Bengio, Y. and Courville, A. [2016]. *Deep Learning*, MIT Press. http://www.deeplearningbook.org.

[43] Goodfellow, I. J., Shlens, J. and Szegedy, C. [2014]. Explaining and harnessing adversarial examples, *arXiv preprint arXiv:1412.6572* .

[44] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [2020]. Generative adversarial networks, *Commun. ACM* **63**(11): 139â144.
URL: *https://doi.org/10.1145/3422622*

[45] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [n.d.]. Generative adversarial nets, *Advances in neural information processing systems* **27**.

[46] Goyal, Y., Khot, T., Summers-Stay, D., Batra, D. and Parikh, D. [2017]. Making the v in vqa matter: Elevating the role of image understanding in visual question answering, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913.

[47] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J. et al. [2018]. Recent advances in convolutional neural networks, *Pattern recognition* **77**: 354–377.

[48] Gu, Q., Wang, Q., Xiong, N. N., Jiang, S. and Chen, L. [2021]. Surrogate-assisted evolutionary algorithm for expensive constrained multi-objective discrete optimization problems, *Complex & Intelligent Systems* pp. 1–20.

[49] Guerriero, A., Pietrantuono, R. and Russo, S. [2021]. Operation is the hardest teacher: estimating dnn accuracy looking for mispredictions, *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, pp. 348–358.

[50] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F. and Pedreschi, D. [2018]. A survey of methods for explaining black box models, *ACM computing surveys (CSUR)* **51**(5): 1–42.

[51] Guo, J., Jiang, Y., Zhao, Y., Chen, Q. and Sun, J. [2018]. Dlfuzz: differential fuzzing testing of deep learning systems, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, ACM, pp. 739–743.
URL: *https://doi.org/10.1145/3236024.3264835*

[52] Haq, F. U., Shin, D. and Briand, L. [2022]. Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization, *Proceedings of the 44th international conference on software engineering*, pp. 811–822.

[53] Haq, F. U., Shin, D., Nejati, S. and Briand, L. [2021]. Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems, *Empirical Software Engineering* **26**(5): 90.

[54] Harman, M. and McMinn, P. [2010]. A theoretical and empirical study of search-based testing: Local, global, and hybrid search, *IEEE Transactions on Software Engineering* **36**(2): 226–247.

[55] Harman, M., McMinn, P., Teixeira de Souza, J. and Yoo, S. [2010]. Search based software engineering: Techniques, taxonomy, tutorial, *in* B. Meyer and M. Nordio (eds), *Empirical Software Engineering and Verification - International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*, Vol. 7007 of *Lecture Notes in Computer Science*, Springer, pp. 1–59.
URL: *https://doi.org/10.1007/978-3-642-25231-0_1*

[56] Hauer, F., Pretschner, A. and Holzmüller, B. [2019]. Fitness functions for testing automated and autonomous driving systems, *Computer Safety, Reliability, and Security - 38th International Conference, SAFECOMP 2019, Turku, Finland, September 11-13, 2019, Proceedings*, Vol. 11698, Springer, pp. 69–84.

[57] He, K., Zhang, X., Ren, S. and Sun, J. [2016]. Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

[58] Hinton, G. E. and Roweis, S. [2002]. Stochastic neighbor embedding, *Advances in neural information processing systems* **15**.

[59] Hu, M. and Liu, B. [2004]. Opinion lexicon, https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html.

[60] Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A. and Tonella, P. [2020]. Taxonomy of real faults in deep learning systems, *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, Association for Computing Machinery, p. 1110â1121.
URL: *https://doi.org/10.1145/3377811.3380395*

[61] Humbatova, N., Jahangirova, G. and Tonella, P. [2021]. Deepcrime: Mutation testing of deep learning systems based on real faults, *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*.

[62] Humeniuk, D., Khomh, F. and Antoniol, G. [2022]. A search-based framework for automatic generation of testing environments for cyber–physical systems, *Information and Software Technology* **149**: 106936.

[63] Jahangirova, G., Stocco, A. and Tonella, P. [2021]. Quality metrics and oracles for autonomous vehicles testing, *Proceedings of 14th IEEE International Conference on Software Testing, Verification and Validation*, ICST '21, IEEE, pp. 194–204.

[64] Jahangirova, G. and Tonella, P. [2020]. An empirical evaluation of mutation operators for deep learning systems, *IEEE International Conference on Software Testing, Verification and Validation*, ICST'20, IEEE, p. 12 pages.

[65] Jogin, M., Madhulika, M., Divya, G., Meghana, R., Apoorva, S. et al. [2018]. Feature extraction using convolution neural networks (cnn) and deep learning, *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*, IEEE, pp. 2319–2323.

[66] Kang, S., Feldt, R. and Yoo, S. [2020a]. Sinvad: Search-based image space navigation for dnn image classifier test input generation, *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pp. 521–528.

[67] Kang, S., Feldt, R. and Yoo, S. [2020b]. SINVAD: search-based image space navigation for DNN image classifier test input generation, *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*, ACM, pp. 521–528.
URL: *https://doi.org/10.1145/3387940.3391456*

[68] Kang, S., Feldt, R. and Yoo, S. [2023]. Deceiving humans and machines alike: Search-based test input generation for dnns using variational autoencoders, *ACM Transactions on Software Engineering and Methodology* .

[69] Khatiri, S., Saurabh, P., Zimmermann, T., Munasinghe, C., Birchler, C. and Panichella, S. [2024]. SBFT tool competition 2024 - cps-uav test case generation track, *IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT@ICSE 2024*.

[70] Kim, J., Feldt, R. and Yoo, S. [2019]. Guiding deep learning system testing using surprise adequacy, *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, IEEE / ACM, pp. 1039–1049.
URL: *https://doi.org/10.1109/ICSE.2019.00108*

[71] King, R. S. [2015]. *Cluster analysis and data mining: An introduction*, Mercury Learning and Information.

[72] Kingma, D. P. and Welling, M. [2013]. Auto-encoding variational bayes, *arXiv preprint arXiv:1312.6114* .

[73] Kirch, W. (ed.) [2008]. *Pearson's Correlation Coefficient*, Springer Netherlands, pp. 1090–1091.
URL: *https://doi.org/10.1007/978-1-4020-5614-7*$_2$569

[74] Klaise, J., Looveren, A. V., Vacanti, G. and Coca, A. [2021]. Alibi explain: Algorithms for explaining machine learning models, *Journal of Machine Learning Research* **22**(181): 1–7.
URL: *http://jmlr.org/papers/v22/21-0017.html*

[75] Kong, Z., Guo, J., Li, A. and Liu, C. [2020]. Physgan: Generating physical-world-resilient adversarial examples for autonomous driving, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14254–14263.

[76] Krause, E. F. [1986]. *Taxicab geometry: An adventure in non-Euclidean geometry*, Courier Corporation.

[77] Kurakin, A., Goodfellow, I. J. and Bengio, S. [2018]. Adversarial examples in the physical world, *Artificial intelligence safety and security*, Chapman and Hall/CRC, pp. 99–112.

[78] Lakhotia, K., Harman, M. and McMinn, P. [2007]. A multi-objective approach to search-based test data generation, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, ACM, pp. 1098–1105.
URL: *http://doi.acm.org/10.1145/1276958.1277175*

[79] Larman, C. [1997]. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall.

[80] Le, Q. and Mikolov, T. [2014]. Distributed representations of sentences and documents, *International conference on machine learning*, PMLR, pp. 1188–1196.

[81] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. [1998]. Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11): 2278–2324.

[82] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. et al. [1998]. Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11): 2278–2324.

[83] Ledel, B. and Herbold, S. [2022]. Studying the explanations for the automated prediction of bug and non-bug issues using lime and shap, *arXiv preprint arXiv:2209.07623* .

[84] Lehman, J. and Stanley, K. O. [2011a]. Abandoning objectives: Evolution through the search for novelty alone, *Evolutionary Computation* **19**(2): 189–223.
URL: *https://doi.org/10.1162/EVCO*$_{a0}$0025

[85] Lehman, J. and Stanley, K. O. [2011b]. Evolving a diversity of virtual creatures through novelty search and local competition, *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, ACM, pp. 211–218.
URL: *http://doi.acm.org/10.1145/2001576.2001606*

[86] Levenshtein, V. I. et al. [1966]. Binary codes capable of correcting deletions, insertions, and reversals, *Soviet physics doklady*, Vol. 10, Soviet Union, pp. 707–710.

[87] Li, Z., Ma, X., Xu, C., Cao, C., Xu, J. and Lü, J. [2019]. Boosting operational dnn testing efficiency through conditioning, *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 499–509.

[88] Linares-Vásquez, M., Li, B., Vendome, C. and Poshyvanyk, D. [2016]. Documenting database usages and schema constraints in database-centric applications, *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 270–281.

[89] Lundberg, S. M. and Lee, S.-I. [2017]. A unified approach to interpreting model predictions, *Advances in neural information processing systems* **30**.

[90] Ma, L., Juefei-Xu, F., Xue, M., Li, B., Li, L., Liu, Y. and Zhao, J. [2019]. DeepCT: Tomographic combinatorial testing for deep learning systems, *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, IEEE, pp. 614–618.
URL: *https://doi.org/10.1109/SANER.2019.8668044*

[91] Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J. and Wang, Y. [2018]. Deepgauge: Multi-granularity testing criteria for deep learning systems, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, ACM, pp. 120–131.
URL: *http://doi.acm.org/10.1145/3238147.3238202*

[92] Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y. and Potts, C. [2011]. Learning word vectors for sentiment analysis, *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150.

[93] MacQueen, J. [1967]. Classification and analysis of multivariate observations, *5th Berkeley Symp. Math. Statist. Probability*, pp. 281–297.

[94] Magesh, P. R., Myloth, R. D. and Tom, R. J. [2020]. An explainable machine learning model for early detection of parkinson's disease using lime on datscan imagery, *Computers in Biology and Medicine* **126**: 104041.

[95] Manning, C. D., Raghavan, P. and Schütze, H. [2008]. *Introduction to Information Retrieval*, Cambridge University Press.

[96] Mao, K., Harman, M. and Jia, Y. [2016]. Sapienz: Multi-objective automated testing for android applications, *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, ACM, pp. 94–105.
URL: *http://doi.acm.org/10.1145/2931037.2931054*

[97] Marculescu, B., Feldt, R. and Torkar, R. [2016]. Using exploration focused techniques to augment search-based software testing: An experimental evaluation, *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 69–79.

[98] McMinn, P. [2004]. Search-based software test data generation: a survey, *Software testing, Verification and reliability* **14**(2): 105–156.

[99] Miller, T. [2019]. Explanation in artificial intelligence: Insights from the social sciences, *Artificial intelligence* **267**: 1–38.

[100] Montavon, G., Lapuschkin, S., Binder, A., Samek, W. and Müller, K.-R. [2017]. Explaining nonlinear classification decisions with deep taylor decomposition, *Pattern recognition* **65**: 211–222.

[101] Mouret, J.-B. and Clune, J. [2015]. Illuminating search spaces by mapping elites.

[102] Musleh, D., Alotaibi, M., Alhaidari, F., Rahman, A. and Mohammad, R. M. [2023]. Intrusion detection system using feature extraction with machine learning algorithms in iot, *Journal of Sensor and Actuator Networks* **12**(2): 29.

[103] Neelofar, N. and Aleti, A. [2023]. Towards reliable ai: Adequacy metrics for ensuring the quality of system-level testing of autonomous vehicles, *arXiv preprint arXiv:2311.08049* .

[104] Nejati, S., Sorokin, L., Safin, D., Formica, F., Mahboob, M. M. and Menghi, C. [2023]. Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial adas and simulink models, *Information and Software Technology* p. 107286.

[105] Nguyen, V., Huber, S. and Gambi, A. [2021]. Salvo: Automated generation of diversified tests for self-driving cars from existing maps, *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*, IEEE, pp. 128–135.

[106] Nichol, A., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I. and Chen, M. [2021]. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, *arXiv preprint arXiv:2112.10741* .

[107] Omernick, M. and Chollet, F. [2020]. Text classification from scratch, `https://github.com/keras-team/keras-io/blob/master/examples/nlp/text_classification_from_scratch.py`.

[108] O'Shaughnessy, M., Canal, G., Connor, M., Rozell, C. and Davenport, M. [2020]. Generative causal explanations of black-box classifiers, *Advances in neural information processing systems* **33**: 5453–5467.

[109] Panichella, A., Kifetew, F. M. and Tonella, P. [2018]. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets, *IEEE Transactions on Software Engineering* **44**(2): 122–158.

[110] Panichella, S., Gambi, A., Zampetti, F. and Riccio, V. [2021]. Sbst tool competition 2021, *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*, pp. 20–27.

[111] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B. and Swami, A. [2016]. The limitations of deep learning in adversarial settings, *2016 IEEE European symposium on security and privacy (EuroS&P)*, IEEE, pp. 372–387.

[112] Pei, K., Cao, Y., Yang, J. and Jana, S. [2019]. Deepxplore: Automated whitebox testing of deep learning systems, *Commun. ACM* **62**(11): 137?145.
URL: *https://doi.org/10.1145/3361566*

[113] Pettersson, E., Megyesi, B. and Nivre, J. [2013]. Normalisation of historical text using context-sensitive weighted levenshtein distance and compound splitting, *Proceedings of the 19th Nordic conference of computational linguistics (Nodalida 2013)*, pp. 163–179.

[114] Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Gupta, B. B., Chen, X. and Wang, X. [2021]. A survey of deep active learning, *ACM computing surveys (CSUR)* **54**(9): 1–40.

[115] Ribeiro, M. T., Singh, S. and Guestrin, C. [2016]. "why should i trust you?": Explaining the predictions of any classifier, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, Association for Computing Machinery, New York, NY, USA, p. 1135â1144.
URL: *https://doi.org/10.1145/2939672.2939778*

[116] Ribeiro, M. T., Singh, S. and Guestrin, C. [2017]. Lime, `https://github.com/marcotcr/lime`.

[117] Riccio, V., Humbatova, N., Jahangirova, G. and Tonella, P. [2021]. Deepmetis: Augmenting a deep learning test set to increase its mutation score, *arXiv preprint arXiv:2109.07514* .

[118] Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M. and Tonella, P. [2020]. Testing machine learning based systems: a systematic mapping, *Empir. Softw. Eng.* **25**(6): 5193–5254.
URL: *https://doi.org/10.1007/s10664-020-09881-0*

[119] Riccio, V. and Tonella, P. [2020]. Model-based exploration of the frontier of behaviours for deep learning system testing, *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE '20, Association for Computing Machinery, p. 13 pages.

[120] Riccio, V. and Tonella, P. [2023]. When and why test generators for deep learning produce invalid inputs: an empirical study, *Proceedings of the IEEE/ACM International Conference on Software Engineering*, ICSE '23, IEEE/ACM.

[121] Rokach, L. and Maimon, O. [2005]. Top-down induction of decision trees classifiers-a survey, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **35**(4).

[122] Rombach, R., Blattmann, A., Lorenz, D., Esser, P. and Ommer, B. [2022]. High-resolution image synthesis with latent diffusion models, *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695.

[123] Ronneberger, O., Fischer, P. and Brox, T. [2015]. U-net: Convolutional networks for biomedical image segmentation, *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, pp. 234–241.

[124] Rousseeuw, P. J. [1987]. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *Journal of computational and applied mathematics* **20**: 53–65.

[125] Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M. and Aberman, K. [2023]. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22500–22510.

[126] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. et al. [2015]. Imagenet large scale visual recognition challenge, *International journal of computer vision* **115**: 211–252.

[127] Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J. and Müller, K.-R. [2021]. Explaining deep neural networks and beyond: A review of methods and applications, *Proceedings of the IEEE* **109**(3): 247–278.

[128] Seaman, C. B. [1999]. Qualitative methods in empirical studies of software engineering, *IEEE Transactions on Software Engineering* **25**: 557–572.

[129] Selinger, P. [2003]. Potrace: a polygon-based tracing algorithm.
URL: *http://potrace.sourceforge.net/potrace.pdf*

[130] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D. [2017]. Grad-cam: Visual explanations from deep networks via gradient-based localization, *Proceedings of the IEEE international conference on computer vision*, pp. 618–626.

[131] Simonyan, K. and Zisserman, A. [2014]. Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556* .

[132] Smilkov, D., Thorat, N., Kim, B., Viégas, F. and Wattenberg, M. [2017]. Smoothgrad: removing noise by adding noise, *arXiv preprint arXiv:1706.03825* .

[133] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N. and Ganguli, S. [2015]. Deep unsupervised learning using nonequilibrium thermodynamics, *International conference on machine learning*, PMLR, pp. 2256–2265.

[134] Stocco, A., Nunes, P. J., dâAmorim, M. and Tonella, P. [2022]. Thirdeye: Attention maps for safe autonomous driving systems, *Proceedings of 37th IEEE/ACM International Conference on Automated Software Engineering, ASE*, Vol. 22.

[135] Stocco, A., Pulfer, B. and Tonella, P. [2022]. Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems, *IEEE Transactions on Software Engineering* .

[136] Stocco, A., Pulfer, B. and Tonella, P. [2023]. Model vs system level testing of autonomous driving systems: a replication and extension study, *Empirical Software Engineering* **28**(3): 73.

[137] Stocco, A. and Tonella, P. [2020]. Towards anomaly detectors that learn continuously, *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 201–208.

[138] Stocco, A., Weiss, M., Calzana, M. and Tonella, P. [2020]. Misbehaviour prediction for autonomous driving systems, *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, pp. 359–371.

[139] Sundararajan, M., Taly, A. and Yan, Q. [2017]. Axiomatic attribution for deep networks, *International conference on machine learning*, PMLR, pp. 3319–3328.

[140] Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., Guo, S., Li, Y.-F., Ma, L., Xue, Y. and Liu, Y. [2023]. A survey on automated driving system testing: Landscapes and trends, *ACM Trans. Softw. Eng. Methodol.* **32**(5).
URL: *https://doi.org/10.1145/3579642*

[141] Tantithamthavorn, C. and Jiarpakdee, J. [2021]. Monash University. Retrieved 2021-05-17.
URL: *http://xai4se.github.io/*

[142] Tatulli, E. and Hueber, T. [2017]. Feature extraction using multimodal convolutional neural networks for visual speech recognition, *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 2971–2975.

[143] Tian, Y., Pei, K., Jana, S. and Ray, B. [2018]. Deeptest: Automated testing of deep-neural-network-driven autonomous cars, *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, ACM, pp. 303–314.
URL: *http://doi.acm.org/10.1145/3180155.3180220*

[144] Tjoa, E. and Guan, C. [2020]. A survey on explainable artificial intelligence (xai): Toward medical xai, *IEEE transactions on neural networks and learning systems* **32**(11): 4793–4813.

[145] Tzerpos, V. and Holt, R. C. [1999]. Mojo: A distance metric for software clusterings, *Sixth Working Conference on Reverse Engineering, WCRE '99, Atlanta, Georgia, USA, October 6-8, 1999*, p. 187.

[146] Utting, M., Pretschner, A. and Legeard, B. [2012]. A taxonomy of model-based testing approaches, *Software testing, verification and reliability* **22**(5): 297–312.

[147] Van der Maaten, L. and Hinton, G. [2008]. Visualizing data using t-sne., *Journal of machine learning research* **9**(11).

[148] Vidoni, M. [2021]. Evaluating unit testing practices in r packages, *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, pp. 1523–1534.

[149] Wang, J. and Dong, Y. [2020]. Measurement of text similarity: a survey, *Information* **11**(9): 421.

[150] Weiss, K., Khoshgoftaar, T. M. and Wang, D. [2016]. A survey of transfer learning, *Journal of Big data* **3**(1): 1–40.

[151] Whitley, L. D. [1989]. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, *in* J. D. Schaffer (ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989*, Morgan Kaufmann, pp. 116–123.

[152] Wu, Z., Wang, Z., Chen, J., You, H., Yan, M. and Wang, L. [2024]. Stratified random sampling for neural network test input selection, *Information and Software Technology* **165**: 107331.

[153] Xiang, Y., Huang, H., Li, S., Li, M., Luo, C. and Yang, X. [2023]. Automated test suite generation for software product lines based on quality-diversity optimization, *ACM Transactions on Software Engineering and Methodology* **33**(2): 1–52.

[154] Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J. and See, S. [2019]. Deephunter: A coverage-guided fuzz testing framework for deep neural networks, *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, Association for Computing Machinery, pp. 146–157.
URL: *https://doi.org/10.1145/3293882.3330579*

[155] Zhang, J. M., Harman, M., Ma, L. and Liu, Y. [2020]. Machine learning testing: Survey, landscapes and horizons, *IEEE Transactions on Software Engineering* **48**(1): 1–36.

[156] Zhang, M., Zhang, Y., Zhang, L., Liu, C. and Khurshid, S. [2018]. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, ACM, pp. 132–142.
URL: *https://doi.org/10.1145/3238147.3238187*

[157] Zhang, X., Xie, X., Ma, L., Du, X., Hu, Q., Liu, Y., Zhao, J. and Meng, S. [2020]. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty, *Proceedings of 42nd International Conference on Software Engineering*, ICSE '20, ACM, p. 12 pages.

[158] Zhang, Y., Xing, J., Lo, E. and Jia, J. [2023]. Real-world image variation by aligning diffusion inversion chain, *arXiv preprint arXiv:2305.18729* .

[159] Zohdinasab, T., Riccio, V., Gambi, A. and Tonella, P. [2021a]. Deephyperion: exploring the feature space of deep learning-based systems through illumination search, *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 79–90.

[160] Zohdinasab, T., Riccio, V., Gambi, A. and Tonella, P. [2021b]. :replication package, `https://github.com/testingautomated-usi/DeepHyperion`.

[161] Zohdinasab, T., Riccio, V., Gambi, A. and Tonella, P. [2022]. Efficient and effective feature space exploration for testing deep learning systems, *ACM Trans. Softw. Eng. Methodol.* . Just Accepted.
URL: *https://doi.org/10.1145/3544792*

[162] Zohdinasab, T., Riccio, V. and Tonella, P. [2023a]. An empirical study on low-and high-level explanations of deep learning misbehaviours, *2023 ACM/IEEE Interna-*

tional Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, pp. 1–11.

[163] Zohdinasab, T., Riccio, V. and Tonella, P. [2023b]. :replication package, `https://github.com/testingautomated-usi/DeepAtash`.

[164] Zohdinasab, T., Riccio, V. and Tonella, P. [2023c]. Unboxer:replication package, `https://github.com/testingautomated-usi/unboxer`.

[165] Zohdinasab, T., Riccio, V. and Tonella, P. [2024]. :replication package, `https://github.com/testingautomated-usi/DeepTheia`.